DEVELOPING A HYBRID VIRTUALIZATION PLATFORM DESIGN FOR CYBER WARFARE TRAINING AND EDUCATION

THESIS

Kyle E. Stewart
2nd Lieutenant, USAF

AFIT/GCE/ENG/10-06

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/10-06

DEVELOPING A HYBRID VIRTUALIZATION PLATFORM DESIGN FOR CYBER WARFARE TRAINING AND EDUCATION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Kyle E. Stewart
2nd Lieutenant, USAF

June 2010

AFIT/GCE/ENG/10-06

DEVELOPING A HYBRID VIRTUALIZATION PLATFORM DESIGN FOR CYBER WARFARE TRAINING AND EDUCATION

Kyle E. Stewart, B.S. Computer Engineering
2nd Lieutenant, USAF

Approved:

//signed//	01 Jun 2010
Lt Col Jeffrey W. Humphries (Chairman)	
//signed//	01 Jun 2010
Dr. Gilbert L. Peterson (Member)	
//signed//	01 Jun 2010
Dr. Michael R. Grimaila (Member)	

Abstract

Virtualization is a technique used to model and simulate the cyber domain, as well as train and educate. Different types of virtualization techniques exist that each support a unique set of benefits and requirements. This research proposes a novel design that incorporates host and network virtualization concepts for a cyber warfare training platform. At the host level, hybrid virtualization combines full and operating system virtualization techniques in order to leverage the benefits and minimize the drawbacks of each individual technique. Network virtualization allows virtual machines to connect in flexible topologies, but it also incurs additional processing overhead.

Quantitative analysis falls into two sets of experiments. The first set of experiments evaluates traditional virtualization techniques against the hybrid approach. Results indicate that in some cases, performance of hybrid virtualization exceeds that of full virtualization alone while still providing an identical feature set. The second set of experiments examines the amount of overhead involved with network virtualization with respect to bandwidth and latency. Results indicate that performance over a local area network incurs two to four times the performance cost compared to physical connections. The benefit of this additional overhead is an increased flexibility in defining network topologies at the software level independent of the underlying physical topology.

Acknowledgments

I would like to give acknowledgment to my thesis advisor Maj Todd Andel for both the academic and professional development assistance he has provided. He has literally made this degree possible for me and has shown great dedication to helping me succeed. I would also like to thank my family for the support they have given me throughout this experience.

Kyle E. Stewart

Table of Contents

List of Figures

Figure Page

List of Tables

DEVELOPING A HYBRID VIRTUALIZATION PLATFORM DESIGN FOR CYBER WARFARE TRAINING AND EDUCATION

I. Introduction

> Know your enemy and know yourself; in a hundred battles you will never be in peril. . . . If ignorant of your enemy and of yourself you are certain in every battle to be in peril. -Sun Tzu

Over the last few decades, the worlds of computer networks and information security have converged to create a fast moving and incredibly dynamic warfighting domain. United States Air Force leaders have recognized the importance of training and equipping airmen to effectively fight and win in this modern battlefield [1]. In a joint Letter to Airmen, Air Force Secretary Michael B. Donley and Air Force Chief of Staff Gen. Norton Schwartz state that the fight to secure cyberspace is vital to current conflicts as well as a critical component to maintaining a technological advantage over future adversaries. They conclude the letter by stating that all Airmen share responsibility to fight in this "mission-critical domain" so that the broader Air Force mission can be carried out. [2]

1.1. Research Motivation

1.1.1 In order to carry out the charge by Air Force leadership to train Airmen to fight effectively in the cyber domain, it is important that the proper technologies exist to

allow for realistic training environments. Training warriors in realistic battlefield environments has long been a tenet of military practice. Undergraduate pilots spend hours in flight simulators designed to recreate the instrumentation and performance characteristics of their aircraft. The Army has invested funds into realistic gaming environments to increase the ability of soldiers to work cooperatively in teams to prepare them for actual combat situations [3]. The cyber domain is no different. If cyber warriors wish to attack, defend and exploit information systems it is critical that they have realistic environments in which to conduct training. This research is focused on what technologies exist to create the realistic training in the cyber domain.

1.2. Overview and Goals

1.2.1 There are essentially two predominant methods to modeling the cyber domain in order to conduct training: duplication and virtualization. Duplication refers to the physical duplication of operational equipment for the purposes of training. Although this provides a very close approximation to actual operational conditions, the monetary, time and manpower investments can limit the number of training ranges available to Airmen [4]. The other method is virtualization. Virtualization allows some types of operational networking equipment (such as desktops, servers, routers, switches, hubs) to be simulated inside a computer system. Since the computing requirements of some of these components are far less than the computation capability of modern systems, a virtual environment can provide an efficient method of recreating virtualized computer networks on a smaller set of physical machines [5, 6, 7, 8].

1.2.2 Many education institutions that teach courses in network security and computer system administration have leveraged virtualization technology to provide hands-on laboratories for their students [9, 6, 10, 11, 12]. However, there exist a wide variety of virtualization techniques for enabling these types of research laboratories. Each technique comes with its own set of strengths and weaknesses. In general terms, virtualization techniques fall into two broad categories. Some techniques can provide high density at the cost of platform flexibility. In other words, the technology supports very *lightweight* virtual machines but they all have to be of the same type. On the other hand, other virtualization technologies provide platform flexibility at the cost of high resource consumption. These *heavyweight* virtual machines can run a variety of operating system types, but require more dedicated resources to accomplish the task. Traditionally, educational institutions that have developed hands-on laboratories based on virtualization technology have chosen one specific virtualization technology over another based on their needs and educational goals [11, 13].

1.2.3 The purpose of this research is to examine ways in which lightweight and heavyweight virtualization may be combined in order to leverage the strengths of both. The research examines which heavyweight and lightweight virtualization technologies are the most compatible and effective when combined on the same physical platform. The research also investigates the performance characteristics of the hybrid virtualization platform when compared against traditional virtualization techniques. This data could then be used in future research to determine which solution to adopt in the development of a training platform.

1.2.4 The research also examines the role of network based virtualization techniques in building a cyber warfare education and training platform. The specific focus of research is on an implementation of a peer to peer virtual private network solution called N2N. This research conducts both latency and bandwidth benchmarks in environments both with and without network virtualization. The experiments show that there is about a two to four times slowdown in the latency and bandwidth connection capability under network virtualization. The benefit of network virtualization is that it allows machines to connect and form arbitrary network topologies regardless of the underlying physical topology. The use of network virtualization depends on the application but if the amount of overhead is tolerable serves as a viable approach for a cyber warfare training platform.

1.3. Thesis Layout

1.3.1 Figure 1 presents a conceptual roadmap for the research presented in this thesis. The problem space is defined through the presentation of the state of the art in virtualization and education. The analysis of virtualization in education leads to a design approach that seeks to improve the state of the art through host and network based virtualization methods. For each of these methods, a set of experiments help to validate the assumptions made in the proposed solution. Each of these parts contribute towards taking the role of virtualization a step forward in developing a training and education platform for cyber warfare.

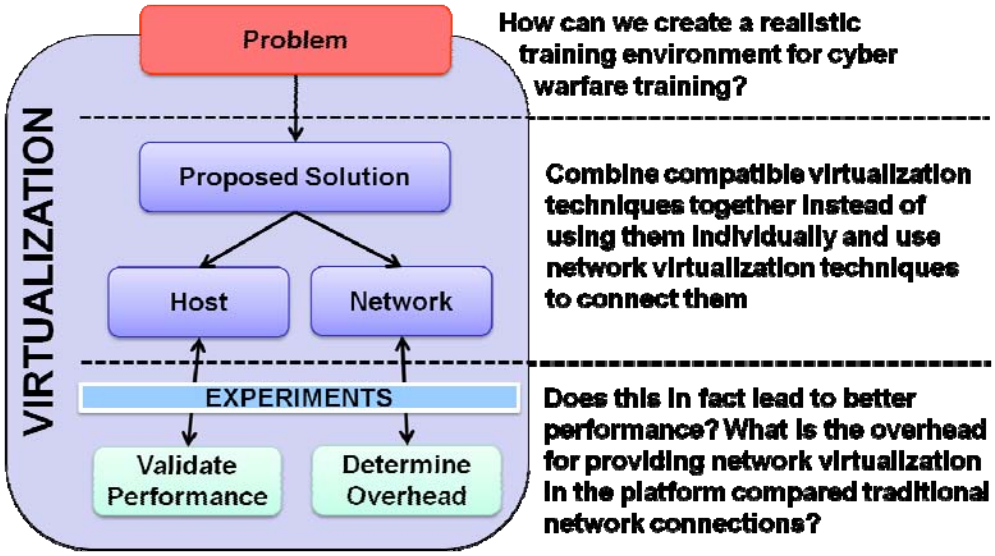

Figure 1: Research roadmap that outlines the layout of this thesis

1.3.2 This thesis is divided into a number of chapters. The following sections provide a summary of the contents of each chapter. Chapter II discusses background research related to virtualization technology. Chapter III analyzes the state of the art in virtualization based network security laboratories and introduces a hybrid virtualization platform that seeks to improve performance over current methods. Chapter IV describes the methodology for characterizing the performance of the hybrid virtualization platform, the results of which are presented in Chapter V. Finally, Chapter VI summarizes the research and provides conclusions and recommendations for future work in this area.

1.3.3 Chapter II *Virtualization Literature Review* provides important background information on key virtualization concepts. These concepts are critical to understanding the research discussed throughout this thesis. The chapter covers the history and issues relating to virtualization on the x86 platform. The chapter discusses key virtualization techniques such as full virtualization, paravirtualization, container-

based virtualization and ported virtualization. The chapter describes the specific technologies that exist for each type of virtualization, with a focus applied to technologies that exist as open source products or are freely available for educational use. Finally, the chapter concludes with a discussion of relevant technologies in network virtualization.

1.3.4 Chapter III *Virtualization in Network Security Education* examines current implementations of virtualization technology used to create hands-on educational or training environments. Although a large number of projects exist throughout academia, the list has been narrowed down to a handful of representative examples. At least one project represents each type of virtualization technology discussed in Chapter II. Finally, the chapter introduces the concept of a hybrid virtualization platform and compares the capabilities of such a platform against traditional virtualization solutions with regards to the requirements established in this chapter.

1.3.5 Chapter IV *Methodology* describes the experimental setups this research uses to determine the performance characteristics of both host and network based virtualization. This chapter outlines the set of experiment parameters, design considerations and hardware for the experiments. The host based experiments run a set of benchmarks inside virtual machines that use traditional virtualization techniques as well as the hybrid technique described in Chapter III. The network virtualization experiments use the network virtualization tool N2N and compares the performance against a baseline physical network connection.

1.3.6 Chapter V *Results* presents the results from the experiments described in Chapter IV. The host based experiments show that in some of the benchmarks,

performance is very similar amongst the different hypervisor platforms. The Compile Apache benchmark shows the greatest amount of difference in performance. In this case the hybrid approach presents performance characteristics that fall in between the performance profiles of its full and operating system virtualization components. This demonstrates that the hybrid approach is capable of performance that exceeds that of full virtualization alone while still providing the capability to support multiple guest operating systems. The network virtualization experiments show that network virtualization incurs an approximate two to three times slowdown in performance relative to direct physical connections.

1.3.7 Chapter VI *Conclusions and Recommendations* provides a final commentary on the research presented in this thesis. Although hybrid virtualization demonstrated the ability to provide improved performance compared to full virtualization alone, it is not without its drawbacks. Practical experience with the platform suggests that other instabilities might be introduced into the system by running two hypervisors on the same system. The data from Chapter V also suggests that many tasks might not experience any performance benefit from this configuration. Instead it is recommended that the granularity of hypervisor diversity remains at the network level and that a hybrid approach focus on the network virtualization component of a cyber warfare training platform. Network virtualization would allow virtual machines from different hypervisors to connect seamlessly. This network approach of hybrid virtualization allows the flexibility of multiple hypervisors and provides the basis for a realistic cyber warfare training and education environment.

II. Virtualization Literature Review

Virtualization technology has been around for several decades now. While its first use dates back to the IBM mainframes of the 1960s, the dramatic increase in performance of desktop computers over the last decade has pushed this technology into mainstream desktop computing [14, 15]. It has also begun to reshape the way data centers and other types of information technology centers manage their network and computing resources. This recent surge of technology innovation in virtualization technology, specifically in the area of desktop computing, has spurred on a variety of interesting applications of virtualization technology [7]. While new and quickly emerging technologies open new doors and create new possibilities, it also becomes difficult to keep up with the rapid pace of development. It is important to carefully examine ways in which virtualization may be correctly utilized in order to support national cyberspace objectives [1].

2.1. Overview

2.1.1 This chapter divides background information into three broad categories that are necessary in order to understand the concepts explained throughout this thesis. The first topic deals with the technical aspects of virtualization technology and examines the wide array of tools and concepts in this domain. The idea of virtualization in computing is broad and there are a wide range of techniques and technologies that have been developed to address different areas of virtualization. This report specifically targets virtualization techniques and technologies in the realm of what is referred to as platform virtualization. Key techniques in this area discussed in this chapter include

paravirtualization, full virtualization and container-based or operating system virtualization.

2.1.2 The second part of this chapter is dedicated to the specific products that are representative of each virtualization technique. The focus of this research is on technologies that are available under open source licenses or that are at least available freely to educational institutions for academic purposes. The purpose of this qualification is to provide a platform that can leverage the knowledge base already available at educational institutions.

2.1.3 The final section provides a brief discussion on the topic of network virtualization. These are frameworks that have been developed to facilitate the creation of large virtual networks or to otherwise control and manage virtualization technology. Although there are large number of techniques available to virtualize the network layer [16, 17, 18, 19], the focus of this research is on peer to peer virtual private networks. This section gives a brief introduction to the concept of virtual private networks and how centralized models differ from peer based or decentralized models.

2.2. Introduction to Virtualization

2.2.1 Virtualization has started to become a commonplace word in modern day information technology circles. The term itself is actually a bit vague as it can be used to describe a very broad range of concepts in computer science. One way to define virtualization is to think of it in terms of abstracting and separating a service request from the physical delivery of that service [20]. Virtualizing something on a computer system refers to taking an object, system or capability and simulating its effect without

necessarily physically replicating the original object. In this sense, as long as the service is provided, the underlying mechanism for providing that service can change. A virtual memory system is one common example of this type of abstraction. Operating systems use hard disk swap space to virtualize the effect of having a full address space of physical memory. Another popular example is when multiple physical storage devices are virtualized (for example by a Redundant Array of Independent Disks or RAID system) and effectively appear to the operating system as one logical drive.

2.3. Definition of Terms

2.3.1 Since virtualization has branched off in several different types of technologies and approaches, the terminology has branched along with it. However, each branch of virtualization tends to share the same core set of concepts albeit in different terms. In order to maintain consistency in this document, the following terms provide a common language to describe the different types of virtualization.

- **Hypervisor**

  Within the context of this research, the term *hypervisor* refers to the primary entity that provides the abstractions necessary for virtualization to occur. The mechanism that provides this abstraction changes from technology to technology. It can exist at the hardware level (hardware assist), at the operating system level (container and full virtualization), or can even be the operating system itself (ported paravirtualization). Some consider hypervisors that execute as the primary control software on a physical machine to be Type I hypervisors or *bare-metal* hypervisors. This is to indicate that the hypervisor runs directly on the CPU. Type

II or *hosted* hypervisors run as an application under the control of a main operating system [21]. In literature, this abstraction mechanism is sometimes referred to as a virtual machine monitor.

- **Kernel**

  The *kernel* is the core scheduling and resource management component of an operating system. The kernel software schedules the processors time amongst the various user processes on the system, manages memory, and arbitrates access to system peripherals. When a computer system is booted, the kernel is one of first pieces of software to run on the system and generally occupies a privileged state on the processor in order to execute its management functionality.

- **Host**

  The *host* refers to the underlying physical hardware system. Typically this refers to a complete computer system including processor, memory, display and any required peripherals. Physical hosts can range from laptops and desktop machines to high performance rack-mounted and clustered servers.

- **Guest**

  The *guest* refers to the virtualized system that runs on top of a physical hardware system. This virtual machine is the abstracted representation of a computer system provided by the hypervisor. The underlying virtualization technology determines what type of guests may run on certain host operating systems and hardware.

2.4. Virtualization Techniques

2.4.1. Types of Virtualization

2.4.1.1 One goal of virtualization is to simulate the effect of an entire computer system. Although this technique goes by a variety of names, this document refers to this process as platform virtualization. A platform in this sense represents a specific computer architecture. Examples of computer platforms include Intel's x86, IBM/Motorola PowerPC, MIPS or ARM platforms. Platform virtualization (of which the x86 platform is perhaps the most popular in the desktop world) falls into roughly three categories: emulation/full virtualization, paravirtualization and operating system virtualization.

2.4.1.2 Emulation generally refers to the process of translating each instruction of the emulated platform to equivalent instructions on the host platform. Since this translation occurs on every instruction, the emulation overhead can be significant. When emulating a platform on top of itself (for example an x86 platform on an x86 processor), the hardware can execute the majority of instructions natively without any translation. This is known as full virtualization and can result in almost near native performance. With paravirtualization, a modified guest operating system kernel communicates directly with the hypervisor in order to minimize the performance penalty of virtualizing system calls. Operating system virtualization uses a shared system kernel to isolate and manage resources in such as way that special user processes can be made to act like independent machines. Since all virtual machines share the same system kernel, this means that all virtual machines must run the same operating system (e.g., Linux on Linux, Windows on

Windows). The following sections provide more detail each of these types of platform virtualization.

2.4.2. Full Virtualization

2.4.2.1 Due to the limitations of the x86 design with regards to virtualization and the low demand for virtual machine technology on the desktop through the 1970s and 1980s, virtualization on the desktop did not progress. During the 1990s, however, desktop hardware became more powerful and underutilized, causing resurgence in research into virtualization as a form of server consolidation [7, 22].

2.4.2.2 In 1998, researchers at Stanford researchers found a way to fully virtualize the x86 platform [20]. The name full virtualization is due to the way that the hypervisor presents a full abstraction of an x86 hardware system to the virtual machine environment. This includes a virtual memory system, virtual CPU, virtual hard disk, virtual console and any other hardware devices. These resources are presented in a way such that the software that executes on top of this abstracted system is generally unaware that the virtual hardware is actually provided by a software hypervisor. The method of dynamic binary translation and direct execution allow the virtual machine to run the majority of code natively without any intervention by the hypervisor. Finally, since the hypervisor presents the entire software interface of virtual hardware to the virtual machine, it is able to mediate all access to physical resources such as CPU, memory and I/O devices. By satisfying these conditions described in Appendix A, the x86 platform became a viable option for virtualization.

2.4.2.3 Dynamic Binary Translation and Direct Execution are two methods that have been developed to get around virtualization problems associated with the original x86 design. When Popek and Goldberg defined the requirements for a processor to support virtualization, they provided two classifications of instructions [5, 23]. Privileged instructions are instructions that require the processor to be in the appropriate privilege level for interacting with hardware. Sensitive instructions are those that affect the state of the hardware (or the hypervisor if the processor is virtualized). In full virtualization, the guest operating system runs at an unprivileged level. When the guest operating system executes a privileged instruction, that instruction causes a security exception to occur. At this point the hypervisor which runs at a higher privilege level steps in and manipulates the virtual hardware to provide the illusion to the guest operating system that it has executed a privileged instruction on real hardware. This intervention process is called a trap. In order to meet the Popek and Goldberg virtualization requirements, all sensitive instructions must trap into the hypervisor [5, 23].

2.4.2.4 The problem on the x86 platform is that not all sensitive x86 instructions are privileged. This means that there are instructions that affect the hardware that do not invoke the security exception process described above. There are in fact 17 such instructions in the x86 instruction set [5]. Dynamic binary translation scans at runtime the code the guest kernel is about to execute, looking for these problem instructions. The code is then dynamically patched with instructions that explicitly call into the hypervisor in order to handle the privileged instruction. Direct execution simply refers to the idea that the vast majority of guest kernel and user code may execute directly on the processor

without any intervention from the hypervisor [14, 23]. This direct execution of instructions is what allows for almost native speed of the guest operating system. Figure 2 provides an illustration of how full virtualization fits in with the host kernel, hardware and guest operating systems.

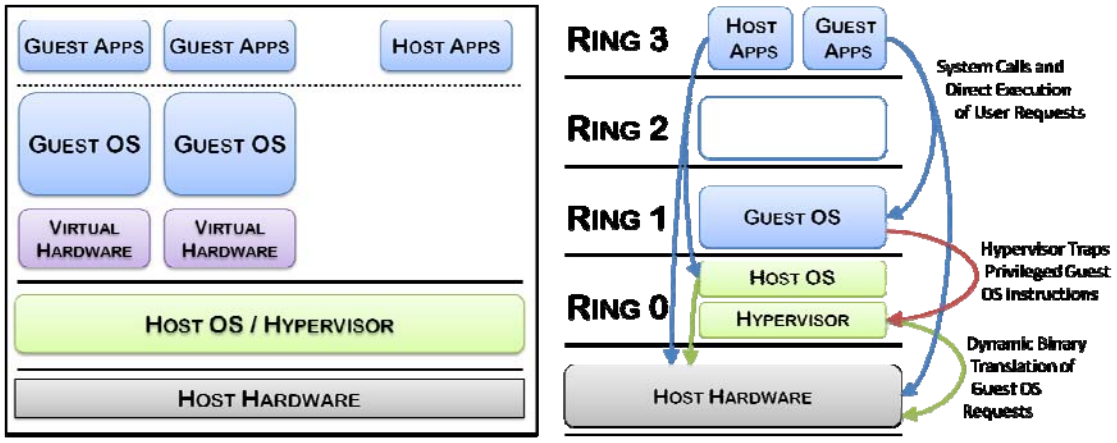

Figure 2: Program execution layout when using full virtualization

2.4.3. Hardware Assist with Virtualization Extensions

2.4.3.1 Due to the high demand for efficient virtualization capability for the x86 platform, both Intel and AMD have developed extensions to the x86 instruction set that satisfy the virtualization requirements established by Popek and Goldberg. The extensions work by providing two additional modes. These are referred to as *root* and *non-root* privilege modes. With hardware assist, the guest operating system runs in the unprivileged non-root mode and the hypervisor runs in the privileged root mode. Each mode has its own set of ring levels 0 through 3. This means the guest operating system code runs at ring 0 in the non-root mode and the hypervisor runs at ring 0 in the root mode. Whenever code in the unprivileged non-root mode attempts to execute privileged

instructions (even code running in non-root ring 0), the hypervisor can now properly trap the instruction and provide the necessary virtualization capability [20].Figure 3 illustrates how the root and non-root privilege levels interact to force guest operating system calls to trap into the hypervisor.

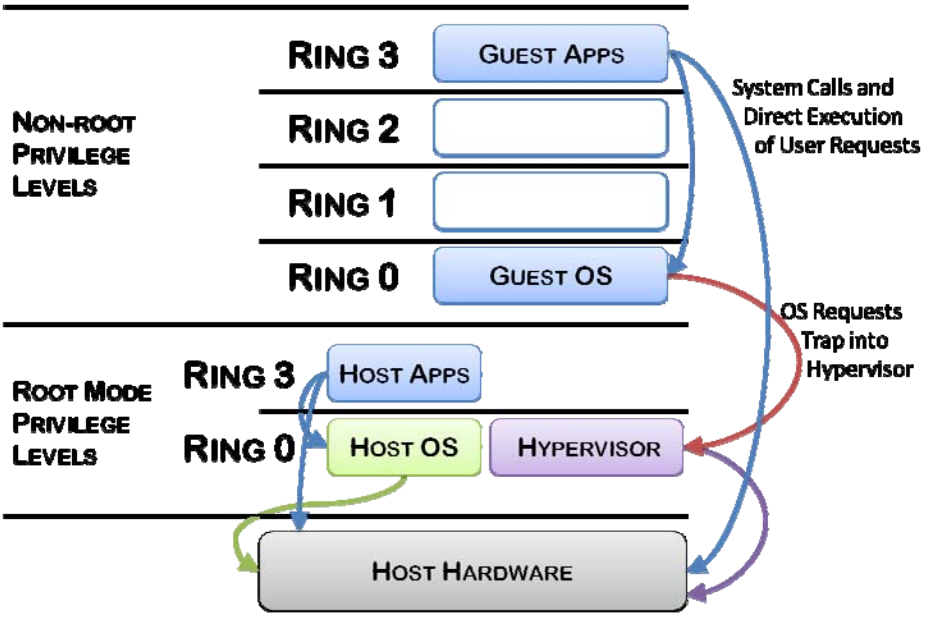

Figure 3: Hardware assist flow of execution

2.4.3.2 When the first generation of the hardware assist feature was released, the programming model forced a significant amount of traps into the root mode hypervisor. The context switch costs that occurred to handle the change in modes introduced significant overhead. The overhead was substantial enough that traditional methods such as dynamic binary translation and direct execution outperformed the virtualization capabilities of hardware assist [20]. As the extensions have matured, the efficiencies in the memory management capabilities and reduction in context switching have improved

performance to the point where hardware assist is a viable method of providing full virtualization capability.

2.4.4. Paravirtualization

2.4.4.1 One of the major drawbacks to the traditional methods used to implement full virtualization is the high overhead costs in processing system calls and trapping problem privileged instructions. Paravirtualization is one method that attempts to reduce this virtualization overhead by integrating the system call process into the virtualization layer [24]. Traditionally this is done by modifying the system call code directly in the guest operating system. These modified system calls are referred to as *hypercalls*. Hypercalls call directly into the virtualization layer, which runs at a higher privilege level than the guest operating system. Additionally on difficult to virtualize platforms such as the x86, privileged instructions in the guest kernel that do not trap properly must also be replaced with hypercalls into the hypervisor. Since these hypercalls are designed specifically to bypass the overhead required to virtualize traditional system calls and privileged instructions, some performance gains can be seen when compared to full virtualization techniques depending on the workload [20]. Figure 4 illustrates where the different components of paravirtualization fit into the x86 security model.

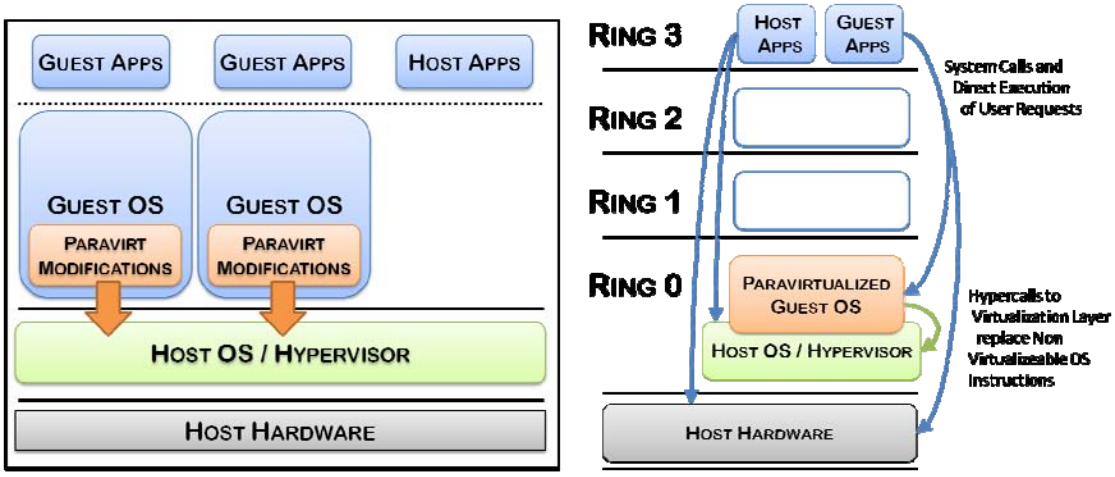

Figure 4: Program execution layout when using traditional paravirtualization

2.4.4.2 There is a unique twist to traditional paravirtualization that focuses on the hardware abstraction layer code of the guest operating system rather than the system call interface. Some operating system designs can be broken down into two main parts. A large top layer contains hardware independent code for performing the operating system responsibilities. A smaller layer of code executes between this top layer and the actual hardware and contains hardware specific code that interfaces the hardware to the rest of the operating system. Although there is not a standardized name for this technique, this document refers to this method of virtualization as ported paravirtualization.

2.4.4.3 The advantage to abstracting out the hardware specific code is that it becomes easier to port the operating systems to run on many types of hardware platforms by only re-implementing the hardware specific code. This is what allows some operating systems like Linux to run on a large variety of hardware platforms. In this type of design, it is possible to port Linux to run on top of a software based system instead of a new hardware system. As long as software exists to virtualize the behavior of the underlying

hardware, the new hardware (and in this case software) abstraction layer can work with the virtualization software to run the rest of the guest operating system unmodified [25].

2.4.5. Container Virtualization

2.4.5.1 Container virtualization is a type of virtualization that provides a high density of guest operating systems by implementing the virtualization layer inside the host kernel [26]. The host kernel then becomes responsible for creating different execution environments that are able to act independent of one another. Each of these execution environments is referred to as a container or sometimes as a virtual private server. In these separate containers is where the guest user applications execute. Each guest container shares the kernel. This provides the ability to dramatically reduce the resource requirements of each container since an entire hardware system does not need to be virtualized and the kernel can more finely control the resources allocated to the containers. Since the host kernel is shared amongst the guest containers, each container must essentially run the same operating system. The kernel is responsible for making sure that each container cannot interfere with the execution of other containers on the same system. Figure 5 illustrates the execution environment that exists under container virtualization along with the mapping to the x86 security model.

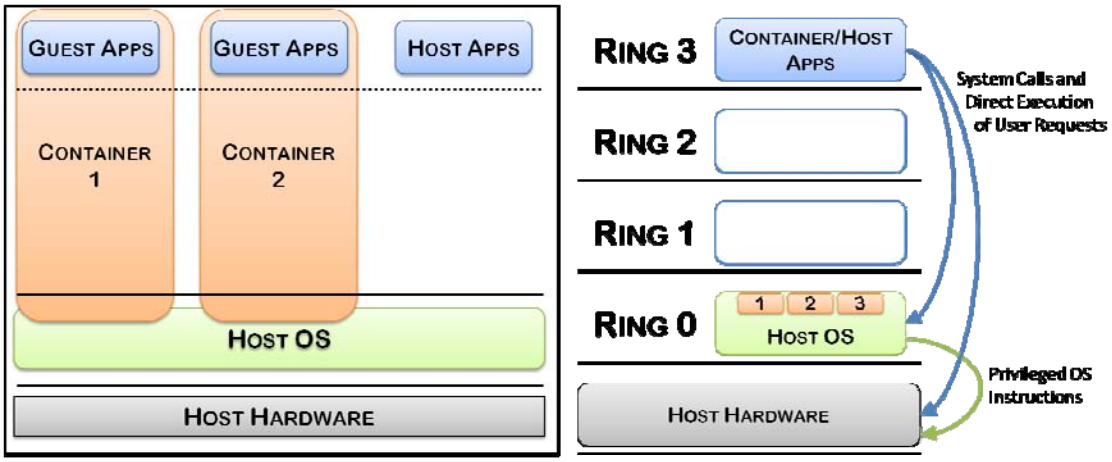

Figure 5: Execution layout for container virtualization

2.5. Virtualization Technologies

2.5.1. Kernel Virtual Machine (KVM)

2.5.1.1 The Kernel Virtual Machine is a hypervisor that has been developed to take advantage of the hardware extensions now available on the x86 platform [27]. It is a loadable module for the Linux kernel that allows the Linux kernel to use the processor virtualization extensions. KVM adds this functionality by running the guest kernel and user level processes in the non-root execution rings. Since it is a module loaded into the Linux kernel, KVM itself runs in the privileged root mode and traps the appropriate instructions from the guest machine. KVM leverages the emulated I/O devices already developed in x86 emulation software QEMU to provide virtual devices such as hard disks and memory to the virtual machine. Since KVM leverages the features found in both the Linux kernel (scheduling, memory management, device drivers) as well as the virtualization capability found in the Hardware Assist extensions, the code base is relatively small (approximately 10,000 lines) [28].

2.5.2. VMware

2.5.2.1 VMware was one of the first companies to bring a successful virtualization product to the x86 desktop market. Today they are a powerful player in both user and enterprise level virtualization products. Much of VMware's technology is built on the concepts of dynamic binary translation and direct execution. Their products vary according to features, cost and support options. The following list summarizes the most important products [20].

- **VMware Player**

  VMware Player is a free application that has the least amount of functionality. It allows users to run VMware virtual machines created with other VMware utilities.

- **VMware Server**

  VMware Server is a free application that provides additional features that VMware Player does not have. Users can create new virtual machines and manage them through a web access system. It installs as an application on top of one of the supported operating systems (Windows or Linux).

- **VMware Workstation**

  VMware Workstation is a commercial (free for educational use) application that is targeted toward creation of virtual machines on the desktop. It supports managed snapshots of virtual machines, the ability to clone virtual machines as well as complex network configurations for connecting multiple virtual machines.

- **VMware ESX/ESXi**

  VMware ESX is a bare-metal hypervisor intended for use in VMware's enterprise

management solutions. It runs directly on the server hardware and most of the management functionality is implemented in VMware's other enterprise software.

2.5.3. VirtualBox

2.5.3.1 VirtualBox is an open-source full virtualization solution, originally developed by the German company Innotek and later acquired by Sun in 2008 [29]. Sun was then later acquired by Oracle in 2009 [30]. VirtualBox relies on the same general techniques as VMware products to provide full virtualization capability. It is capable of software only virtualization through dynamic code recompilation techniques, some of which is based on QEMU source code. VirtualBox is also capable of leveraging virtualization enabled hardware such as Intel VT or AMD-V. VirtualBox is packaged in two different ways. Most of the software is licensed under the GNU Public License and available as open source software. Oracle also maintains a free (for personal and academic evaluation), but closed source version which has a few additional features such as the ability to support USB be devices both locally and remotely. The closed version also provides the capability to manage the machine remotely through the Remote Desktop Protocol [31].

2.5.4. Xen

2.5.4.1 Xen is an open source paravirtualization product licensed under the GNU Public License [24]. Older versions of Xen could only support guest operating systems whose source was available due to the need to add the hypercall interface. This would typically narrow the range of available operating systems to Linux, BSD, Solaris and other UNIX-like operating systems. Recently with the advancement of hardware assist

technology, Xen has been updated to use virtualization extensions to provide support for unmodified guest operating systems through full virtualization techniques.

2.5.4.2 Xen is a modified version of the Linux kernel that runs as a virtualization layer next to the hardware. In Xen terms, operating systems are referred to as domains. The first domain that is created when Xen begins life is called Domain 0 or *Dom0*. This first domain maintains a special privileged state within Xen and is responsible for arbitrating access to all the system devices on the hardware. So it is important the Dom0 system have all the proper drivers for the host hardware. The kernel for the Dom0 domain must be modified to work with the Xen hypervisor which is sitting between the Dom0 domain and the hardware. Once the Dom0 domain is running, additional guest domains (or *DomU* in Xen terms) may be started. DomU guest kernels require different modifications for running on the hypervisor than the DomU domain requires. Their modifications represent the typical paravirtualization modifications described in section 2.4.4.

2.5.5. Linux VServer and OpenVZ

2.5.5.1 Linux VServer and OpenVZ are two popular container virtualization technologies available for the Linux operating system [32, 33]. Both software packages work by modifying the original Linux kernel to add the functionality necessary to allow container virtualization to occur. These modifications enable the strict isolation of different containers in terms of memory allocation, CPU usage and network utilization among other criteria. Both of these software packages also provide userspace utilities that allow the user to manage the containers. Management functionality allows for the fine

grained control of the execution environment of a container. For example, a container may be restricted to only use 5% of the total CPU capability of the host CPU. This becomes useful in network security for training denial of service techniques. A denial of service attack may be launched against a container which from the perspective of the container may consume 100% of the CPU. However, on the host this container is only utilizing 5% of the total CPU capability and so the host is able to continue to execute the other containers at their regular capacity.

2.5.6. User Mode Linux

2.5.6.1 User Mode Linux is a port of the Linux kernel to run on top of itself as the virtualized hardware platform [25]. It allows the Linux kernel to run as a userspace application with Linux operating system acting as a hypervisor. This is considered to be a type of paravirtualization. It is developed and maintained by Jeff Dike and was first documented in 2001. Originally a patch for the Linux kernel, it has since been integrated into the main development tree for recent versions of the Linux kernel.

2.6. Supporting Technologies

2.6.1. Libvirt

2.6.1.1 Libvirt is an application programming interface toolkit for the Linux operating system that allows generic management of different virtualization technologies without the need to customize the code to each type of hypervisor available [34]. Instead it abstracts the general functionality available with virtualization techniques and provides a public coding interface. This public interface connects to a variety of backend

hypervisor drivers that implement the functionality according to the requirements of the desired hypervisor. The following lists the hypervisors supported by Libvirt:

- Xen
- QEMU
- Linux Containers (LXC)
- OpenVZ
- VirtualBox
- OpenNebula
- VMware ESX

2.7. Network Virtualization

2.7.1. Introduction

2.7.1.1 Network virtualization is a method of creating independent network topologies as an additional layer on top of the current underlying network architecture. The public Internet is a popular base network architecture that forms a baseline infrastructure for a wide array of network virtualization techniques. The Internet provides a high-speed, global network due to its large scale adoption and popularity. The architecture of the Internet provides a natural layering approach that allows protocols and applications to function independently of the layer below. Network virtualization works within this layering approach to provide top level applications the ability to work with the network independent of the actual underlying physical topology.

2.7.2. Virtual Distributed Ethernet (VDE)

2.7.2.1 Virtual Distributed Ethernet (VDE) is an abstraction of the networking components involved in a typical Ethernet network [18]. It allows for virtual machines to connect to physical machines in arbitrary network topologies. VDE provides virtual switches and hubs and allows the network adapters of physical machines as well as virtual network adapters of virtual machines to connect to them. Since these networking components are implemented in software, it allows for a great deal of flexibility in implementing arbitrary network topologies for a virtual environment.

2.7.3. Virtual Private Networks

2.7.3.1 Many techniques for providing network virtualization exist. Virtual Private Networking (VPN) is one popular network virtualization technique. The primary purpose of VPNs is to allow the establishment of secure connections between trusted peers on a network. Generally these connections form in such a way as to allow high level network applications to behave as though the other peers in the VPN have connected to the same physical network. In actuality, these peers may be separated by thousands of miles across a complex mesh of networking equipment and interconnection technologies. VPNs usually provide some type of encryption support in order to establish secure tunnels over insecure mediums such as the public Internet [35, 36].

2.7.3.2 Many VPN solutions work in a centralized model [36]. Centralized VPNs provide the ability to centrally control and administer the VPN. Clients that wish to connect to the VPN establish a connection to a VPN server. The VPN server acts as a central location for configuration and administration of the VPN. The client authenticates

to the server and in turn the server may authenticate to the client depending on the security requirements of the VPN. Once the client establishes a secure connection to the VPN server, the VPN server acts as a central point of contact for client communication. In a centralized model, packets destined for other clients must travel through the VPN server in order to properly route through the VPN. As the size the VPN scales, it is necessary that the VPN server has sufficient network bandwidth and computation power to handle the large flow of packets that must traverse the VPN.

2.7.3.3 Some VPNs utilized a decentralized or peer to peer (P2P) model [37, 38, 39]. There are a wide variety of techniques for implementing this type of approach. However, most P2P VPNs share some common characteristics. Generally in a P2P VPN, the VPN clients also play the role of VPN server. The first issue to tackle in a P2P VPN is how to initially discover and connect to other peers in the network. Each P2P VPN solution tends to approach this problem differently. Some approaches such as N2N rely on special peers to keep lists of the peers that are connected to the network [40]. Other techniques rely on techniques such as distributed hash tables borrowed from other P2P technologies [39, 41]. Although discovery approaches differ, the common thread among P2P VPNs is that after discovery the peers make direct connections to each other. This is opposed to the centralized approach where the clients route their traffic through a common central server. A decentralized approach provides the ability to create large, scalable networks that are free from tedious central configuration and administration.

III. Virtualization in Network Security Education

As techniques and technologies have matured over the last two decades, researchers in academia have leveraged virtualization in order to create hands-on laboratories for students in courses in computer administration, network administrator and network security. This first half of this chapter documents various projects that are representative of the major categories of virtualization. Collectively, these projects give insight into the current state of the art with regards to the use of virtualization in teaching computer administration and networking courses. The analysis of current methods gives insight into a solution to improve the state of the art of virtualization in education. This solution is presented in the second half of this chapter. The techniques of hybrid virtualization along with network virtualization via peer to peer virtual private networks form the basis of a proposed platform for conducting cyber warfare education and training. The chapter concludes with the details of the proposed design.

3.1. Full Virtualization Based

3.1.1. Advantages of Full Virtualization in Education

3.1.1.1 The use of full virtualization is a popular option for many educators teaching computer administration courses [9, 10, 13, 42]. Full virtualization gives the flexibility to run multiple types of operating systems. It also allows students to have the flexibility to store their virtual machines on portable storage. This allows students to work in a lab environment without being tied to specific machine. It also allows students to take their virtual machines to their own personal computers, provided they are capable of running resource intensive full virtual machines.

3.1.2. Cyber Defense Exercise (CDX)

3.1.1 The Cyber Defense Exercise (CDX) is an annual cyber warfare event sponsored by the United States National Security Agency (NSA) [43]. The exercise is geared toward the five undergraduate military academies and awards the coveted CDX trophy each year to the team that most successfully defends a custom built network during a one week engagement by NSA attack team personnel. Although not in competition for the trophy, the military graduate institutions Naval Postgraduate School and the Air Force Institute of Technology (AFIT) also participate in the exercise. AFIT dedicates a two quarter course to preparation for the exercise. Full virtualization plays a dominant role in the construction of the AFIT network that needs to support a variety of services including email, instant messaging, web servers and databases. Full virtualization is implemented using VMware Workstation.

3.2. Paravirtualization Based

3.2.1. Types of Paravirtualization Laboratories

3.2.1.1 When flexibility of guest operating system is not a fixed requirement, paravirtualization becomes a very popular choice for researchers building virtual network environments [44, 45]. Mature utilities exist for the scripted creation of paravirtualization based networks that range in size from one node up to several hundred. In the paravirtualization world, there are two major players: Xen and User Mode Linux. Although both products rely on paravirtualization techniques, there are very different with respect to their abilities, requirements, performance and how they have been applied in the creation of virtual networking environments.

3.2.2. Cyber Defense Trainer (CYDEST)

3.2.2.1 CYDEST is a project in active development by ATC-NY as part of a small business initiative grant from Air Force Research Laboratories [46]. It differs from the other projects listed in this research by the fact that its license is not free or open source. It is considered Government Off-the-Shelf. This means while it is not available to the public at large, it is available to any government organization including the Air Force. CYDEST uses both the paravirtualization and full virtualization capabilities of the Xen hypervisor. CYDEST provides training scenarios that gives students the opportunities to explore realistic scenarios involving computer forensics and cyber warfare.

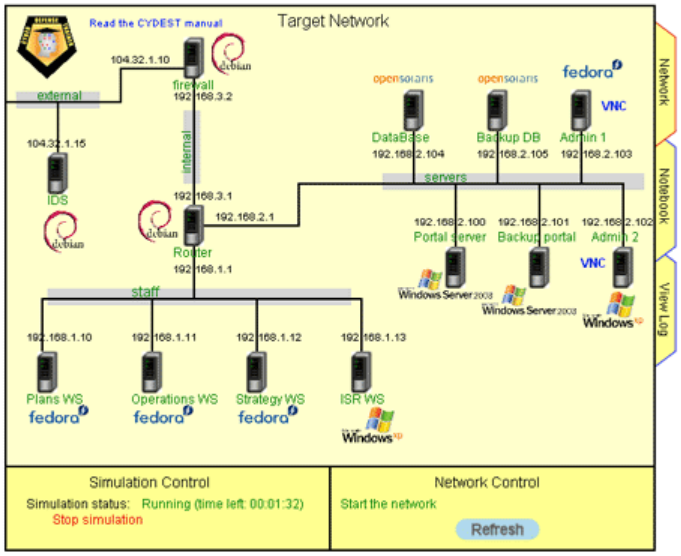

Figure 6: Screenshot showing the network topology for a CYDEST scenario

3.2.3. GINI is not Internet

3.2.3.1 *GINI is not Internet* is a project developed at McGill University in Calgary, Canada [47]. It uses User Mode Linux as the underlying virtualization platform.

GINI. GINI uses a customized UML virtual machine that acts as a common host node. The user can define network topologies graphically using the front end interface tools shown in Figure 7. This creates an XML description of the network that a backend set of software then uses to create the virtual machines and connect the virtual network.

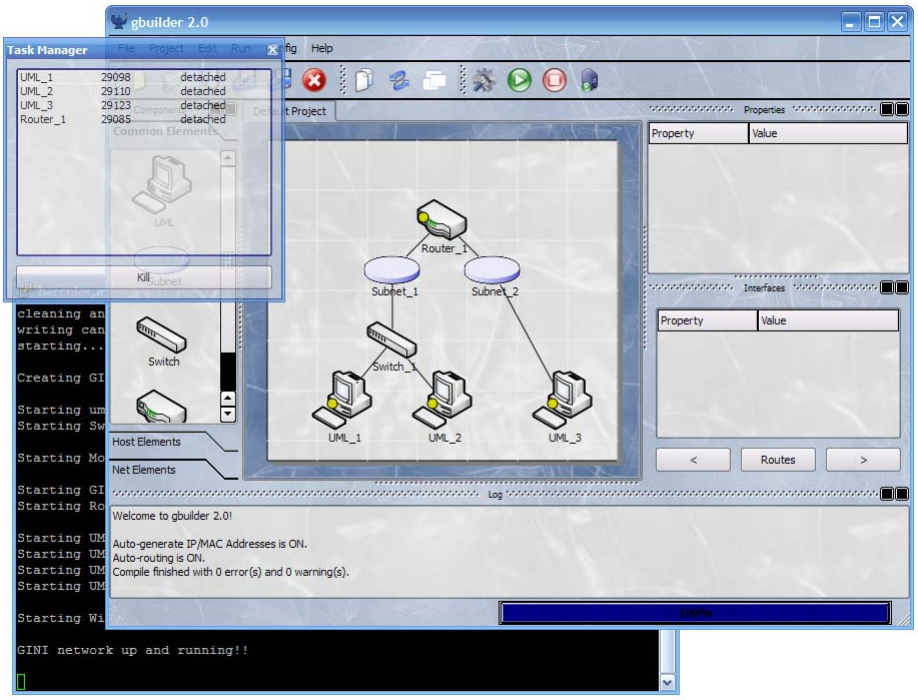

Figure 7: Screenshot showing the network builder tool in GINI

3.3. Container Based

3.3.1.1 Container based labs have not gained as much traction as full virtualization and paravirtualization in the realm of education. This may be due to a perceived higher learning curve, ignorance of container virtualization amongst educators or perhaps the limitation of operating system choices that containers impose. There are systems that demonstrate that container based solutions can form the basis for a virtual

network testbed. The Trellis project is a platform for creating virtual networks on commodity hardware [16]. It uses two types of container technologies Linux VServer and NetNS to form the nodes. It also uses a custom network virtualization system called EGRE to provide connectivity between the virtual machines regardless of the physical host. Some testing with OpenVZ shows that it can provide the same capabilities and performance as the current container technique used in Trellis, but does not integrate with the tunneling mechanism that has been developed.

3.4. Examining Network Laboratories in Education

3.4.1. Network Laboratory Models

3.4.1.1 This section presents a survey of the current work related to creating hands-on networking environments for students taking courses at the undergraduate level. The types of laboratories fall into three main categories: hardware-based labs, decentralized labs, and centralized labs. The following sections examine each of these categories.

3.4.2. Hardware Based Labs

3.4.2.1 Creating a networking lab out of real-world hardware is perhaps the closest one can get to achieving realism. Client machines can be automated to present realistic traffic representing emails, web surfing, file transfer, peer-to-peer networking, and other realistic data. Students have the opportunity to see actual networking hardware, such as routers and switches, and experience the issues involved with cable, power, and space management. Students also have the opportunity to work with proprietary (yet industry standard) network software such as Cisco IOS.

3.4.2.2 Although this type of networking laboratory presents the greatest degree of realism, it also has some serious drawbacks. Perhaps the biggest drawback is cost. Purchasing all the routers, switches, workstations, power supplies, server racks, lab space, cabling, and all the follow- on maintenance costs can be a major setback to introductory courses that lack an ample supply of funding. There is also the cost of time. Often a high level of technical expertise is required to properly configure such a lab which can consume precious time needed by professors and graduate students while they could be spending their time acquiring more funding for their lab. Another drawback is the lack of portability. All the time and effort expended by one team of people to set up a lab does not translate well to another team who wants to replicate the results. Of course, a roadmap has been laid out and some lessons learned can be documented, but the second team still has to put in relatively the same amount of work during implementation as the first team.

3.4.2.3 There are many examples of where this approach has been successful. Georgia Tech's Hands-On Information Security Lab is one example. The infrastructure presents three levels of exploitation difficulty consisting of easy, medium, and hard (represented by an unprotected internet service provider, a university, and a security-conscious internet service provider respectively). This setting allows students to progress up a chain of complexity, using an increasing skill set to solve more challenging problems. The authors of [48] describe how they were able to achieve a certain degree of versatility in the network architecture by exploiting the fact that the Cisco hardware used to connect the nodes worked at both the Layer-2 and Layer-3 portions of the OSI model.

This setup allowed for a virtual rewiring of the routers and switches at the software level that quickly and efficiently creates new network topologies. However, the authors conclude that the laboratory itself exceeded the infrastructure capabilities of many small companies. A consultant from Cisco was even used to help design and implement the network. This type of laboratory, although most likely the closest at achieving realistic network behavior, might exceed the financial resources of many introductory network security classes.

3.4.3. Decentralized Virtualization

3.4.3.1 Many instructors of information security courses realize that the time, energy, cost, and overall difficulty of implementing and maintaining a full-fledged security laboratory is not financially or administratively feasible. These instructors realize that much of the functionality required in such a laboratory could be accomplished by running virtual machines on top of already existing lab infrastructure. Software such as VMware Workstation could be used to produce virtual machines that perform the same functionality as operating systems running on real machines. Furthermore, the inherent networking capabilities built into these packages make them relatively easy to network together when all the host machines are running on the same local area network. Another benefit is that already existing computer labs can be used to build these networks. Virtual machines also allow for a great deal of robustness. Since virtual machine state can be saved, students are free to experiment and potentially break the state of the network. If a student does break the network, the student can quickly revert the virtual machine back to a known working state.

3.4.3.2 There are essentially two models for implementing a decentralized virtual network laboratory. Which model is used depends on the desired capability of the resulting network. In both cases, the distributed computing power and memory capacity of student workstations are used. In the first model, virtual machine images are stored in some sort of centralized storage [49]. When the student wishes to begin a specific lab, the student retrieves the virtual machine images from the central location to the local workstation in the lab. The student then launches these virtual machines on the client machine using the target virtualization platform (such as VMware Workstation for example). One advantage of this approach is that the physical hardware of the host workstations is abstracted away from the virtual machines used in the lab. Another advantage is that each student is working on their own set of virtual machines in an isolated environment, stabilizing the working environment. However, one disadvantage is there can be long delays in copying large virtual machine images across the network for each student's work. Some of this delay can be reduced through the use of linked clones as described in [9]. Another disadvantage is that each network is limited in scale to what the host workstation can individually support (currently around six virtual machines per workstation using full virtualization).

3.4.3.3 The second model provides for larger scale networks at the cost of flexibility [50]. Instead of distributing the same basic set of images to each workstation, a larger set of virtual machine images is distributed among a group of workstations. For example, imagine the need to simulate a network of 30 nodes, but each individual workstation can only run five or six virtual machines simultaneously. The instructor can

divide the network into six parts and give one part to each workstation. This has the advantage of combining resources to create larger networks. This brings us closer to the results of having an actual physical lab. This advantage comes at the cost of complexity and flexibility. If the virtual network has a flat topology, the configuration is rather trivial as each virtual machine is granted direct access to the LAN. More care must be given where broadcast domains within the virtual network must be controlled. Virtual Distributed Ethernet (VDE) can help solve some of this complexity by virtualizing the data link layer and providing a mechanism to connect virtual machines and physical machines to virtual switches and routers [18]. Also, since more than one workstation is used, it might be more difficult to schedule individual time for students to work on lab assignments.

3.4.4. Centralized Virtualization

3.4.4.1 The third model takes a centralized approach to providing a virtual network environment. In this model there is typically a central server that hosts all the virtual networks for all the students. Although in theory this centralized server could support either full virtualization or operating system virtualization, the computing capacity of the server relative to the number of virtual machines that required by the students lends itself to operating system virtualization. Due to the open nature of the Linux operating system and the networking tools available, it is often the platform of choice to deliver this type of virtualization. This has the significant disadvantage that other operating systems such as Microsoft Windows or any of the other BSD and Unix variants cannot be easily integrated into the students' networks. This tradeoff is made for

the advantage that students can log into a central server and create moderately sized networks with minimal impact on resource usage on the central server. This is often combined with methods that allow students to log in from remote locations off campus, allowing a greater amount of freedom and time to work on labs without requiring students to be physically present in computer labs on campus.

3.4.4.2 The central server does not need to be a single machine. A project such as SOFTICE [51] uses the Warewulf cluster software to bind several machines into what appears as a single logical machine. This model provides the advantage of ease of management and allows students easier methods for remote access. In the case of SOFTICE, more computational power can be added by adding more machines to the cluster. However, there is still a student dependence on a central server and the student must have connectivity to this server in order to build and interact with their networks.

3.5. Summary of Virtualization Techniques and Educational Models

3.5.1 Table 1 provides a summary of the different approaches taken to providing virtualization in an education environment. Each approach comes with its own set of advantages and disadvantages that are dictated by the underlying virtualization technology and deployment model. Understanding these tradeoffs and benefits helps to understand the potential of combining techniques in order to minimize the disadvantages of an individual approach.

Table 1: Summary of Virtualization Techniques and Educational Models

Virtualization Category	PROs	CONs
Physical Hardware Model	• Most accurate representation of target environment • Executes at native speed	• Expensive • Requires expert level maintenance • Requires physical accommodations such as power and floor space
Full Virtualization (VMware, VirtualBox)	• Supports multiple guest operating systems • Near native execution speed	• Resource intensive due to full virtualization of memory and other hardware
Paravirtualization (User Mode Linux)	• Does not *require* modification of host system • Support built into the 2.6 line of Linux kernels • Mature set of management utilities	• Slower performance than Xen paravirtualization or container virtualization
Paravirtualization (Xen)	• Tight integration of guest, host and hypervisor leads to performance benefits and reduced overhead in system calls • Open source structure has given Xen good support in the research and academic communities	• Requires source modified guest and host systems which limits support to open source operating systems such as Linux
Container Virtualization (Virtuozzo/OpenVZ, Linux VServer, Solaris Zones)	• Most lightweight and efficient form of virtualization • Scalable	• Requires modification of host operating system
Centralized	• Easy to centrally configure and administer • Easy to support remote connections	• Does not scale well • Need powerful central processing and large storage capacity
Decentralized	• Scales by distributing the computation and storage load to the edges of the network	• Depending on the model, the power of each individual node can determine the amount of virtualization capable in the environment

3.6. The Hybrid Approach – A New Model

3.6.1. Leveraging Multiple Virtualization Technologies

3.6.1.1 Up to this point, all of the network laboratories implemented with virtualization technology have focused on only one type of technology. If the requirements for the course dictates that the laboratory support multiple operating systems or arbitrary operating systems, the laboratory implementers utilize full virtualization (via VMware, Xen, VirtualBox or some other full virtualization solution). If heterogeneous or arbitrary operating system support is not a design requirement, implementers tend to implement the laboratory using paravirtualization (ala Xen or User Mode Linux). If the requirements dictate high density for larger network simulations such as modeling virus and botnet behavior, then implementers tend to use a container based virtualization solution such as OpenVZ [52].

3.6.1.2 The issue with choosing one type of virtualization over another is that along with the strengths of one category of virtualization comes along a set of weaknesses. Full virtualization provides flexibility at the cost of heavy resource usage. Paravirtualization provides slightly better performance at the cost of limited guest operating system availability. Containers provide the best resource allocation for the highest density of guest machines per physical node, but require each container to run the exact same guest kernel which is also shared with the host.

3.6.1.3 Providing the flexibility of multiple types of guest operating systems while simultaneously supporting a higher density of guest machines than traditional full virtualization would support is the basis for the idea of the hybrid virtualization

technique. Hybrid virtualization combines both lightweight and heavyweight virtualization on the same platform in order to leverage the strengths and neutralize the weaknesses of either virtualization technique when used alone.

3.6.2. Example Scenario

3.6.2.1 Consider an example of a network security training scenario illustrated in Figure 8. This network topology represents a typical network that would exist for a small business with both a public Internet facing website as well as an internal intranet website. This network also maintains databases that provide information to the company's websites as well as applications used by the company employees on their workstations. In this example, the company databases hold sensitive corporate information that would be valuable to an attacker.

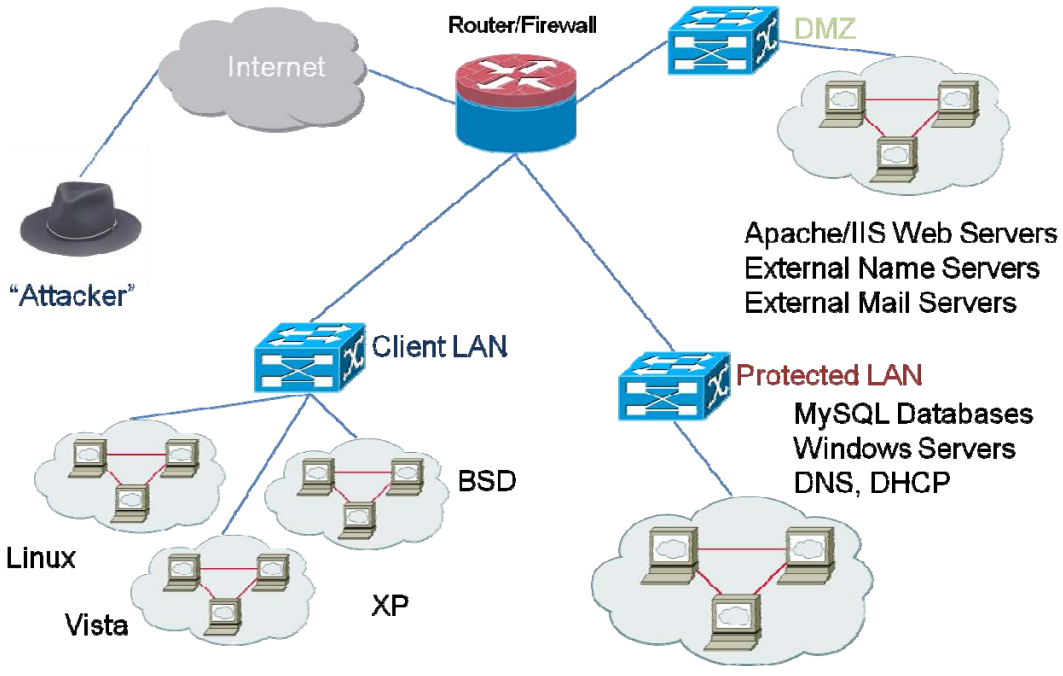

Figure 8: Example realistic network useful for security training

3.6.2.2 A viable attack vector exists through the public facing system running Microsoft Windows Server 2003 and an Internet Information Services (IIS) web server. In order to support this machine in our topology the system must use some flavor of full virtualization. From this machine, an attacker can connect to the databases with the credentials of the web server. The attacker can now also change information on the company intranet web server through an SQL injection technique. Since this attack technique does not necessarily depend on a particular implementation, the system can use a lightweight virtualization such as paravirtualization or container virtualization. Lightweight virtualization can also be used to provide the client workstations with the exception of the database administrator. The database administrator runs Microsoft XP SP2 as their workstation operating system. The attacker must modify the company intranet website to contain malicious code that attacks a known vulnerability in Internet Explorer. This allows the attacker to connect to the database with database administrator credentials and steal the sensitive information stored there.

3.6.2.3 Typically, several hardware nodes would be required in order to represent the scenario described above [46]. There are situations where it might be convenient for a training platform to be able to virtualize the entire scenario on a single machine. For example a student may wish to practice certain destructive attack techniques that could potentially break the expected network behavior. If this system is shared amongst other peers in a classroom environment, this may be problematic for the other students attempting to perform their own attacks. If the student maintained their own offline copy

of the scenario, this would give the student increased flexibility in how they are able to interact with the network.

3.7. Proposed Hybrid Virtualization Platform Design

3.7.1 This section proposes a solution to the problem of creating a network infrastructure. Specifically, the problem is reducing the virtual network footprint to something that the average, modern laptop or desktop can handle. An ideal virtual network would provide the same functionality as an equivalent real-life network. In guiding the virtualization goals, the system must incorporate the type of components that compose real networks. One example is the landscape of what might constitute a typical corporate network. This network most likely runs a large variety of software (from embedded operating system software for routers to large SQL database software in a rack-mount server) on a large number of machines.

3.7.2 This solution seeks to achieve three main objectives. First, the system should support a broad range of unmodified guest operating systems. Second, the system should support a large number of nodes in order to provide the capability to simulate large, more realistic networks. A realistic goal might be 20-30 nodes per 1 GB of RAM available. Finally the system should be portable. To do this, the system uses a building block/blueprint model. In terms of cost of moving a solution around, the building blocks (virtual machine images) are expensive due to their typically large file size. The blueprints should be comparatively much smaller in size and dictate how the building blocks should be put together in order to build our network. With this model, our goal is

to build a large number of possible network “blueprints” with re-useable building blocks. Figure 9 provides a preview of how the layers of our virtualization models fit together.

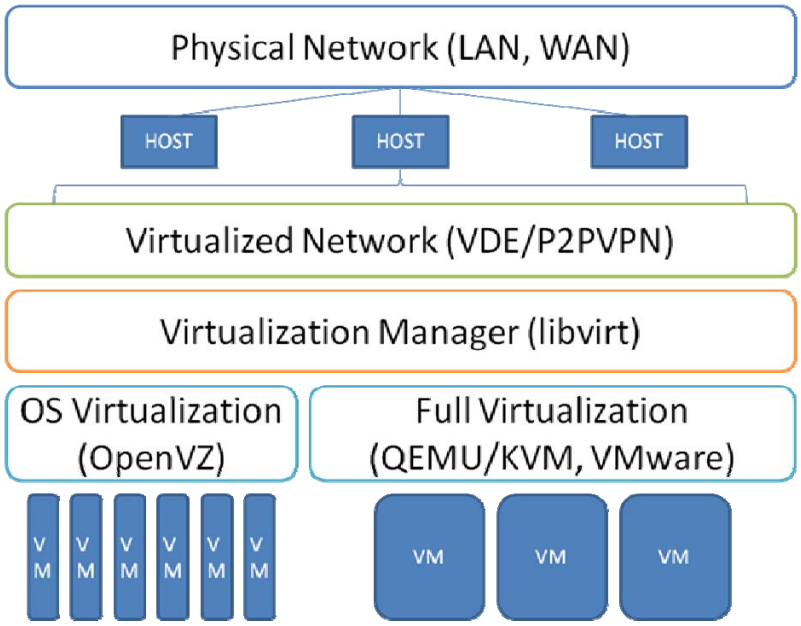

Figure 9: Layers in the virtualization model

3.7.3 Previous solutions have relied on either full virtualization or operating system virtualization. Full virtualization gives us the flexibility to run a large variety of guest operating systems, but prevents us from running a large number of them concurrently due to the inefficient use of system resources (particularly RAM). Operating system virtualization gives us the efficiency to run a large number of nodes per unit of RAM, but requires that all guests run the same operating system kernel as the host.

3.7.4 The solution requires a host operating system that is capable of supporting both full virtualization and operating system virtualization. The Linux kernel provides a convenient open source solution. Modern versions of the Linux kernel have built-in support for KVM hardware-assisted full virtualization. In the case where the host system

does not support the Intel VT or AMD SVM virtualization extensions, KVM automatically falls back to the slower yet still effective emulation capabilities of QEMU. The Linux kernel is also the primary support platform for the OpenVZ project, a container based hypervisor. Both of these projects are being integrated into the mainline Linux kernel and so it is possible that leveraging both of these technologies will become easier. There is already one example of how these technologies can be merged in the ProxMox VE product [53]. ProxMox VE uses both full virtualization and operating system virtualization, demonstrating that the two virtualization techniques can exist side by side.

3.7.5 Another enabling technology that allows these two types of virtualization to be used together is Libvirt. Libvirt is an open source library that provides a common set of application programming interfaces (APIs) to different types of virtualization technologies. Libvirt can act as a driver for both OpenVZ and KVM/QEMU. This means that when in environments that support both of these technologies, the system can use Libvirt as another abstraction layer in order to manage the implementation of each virtual machine.

3.7.6 Another possibility for virtualizing the network layer is the use of peer to peer-virtual private networking (P2P-VPN) technologies to provide a decentralized approach to building virtual networks. In a traditional centralized virtual private network, machines that are connected across a wide area network (WAN) can communicate as if they were connected to the same local area network (LAN), similar to the setups used in SIMTEX [4] and CDX [43]. The edges of the WAN that wish to connect to the VPN

LAN connect to a central server and the clients establish a secure communications channel with the central server. When the client wishes to send a packet to another client on the same VPN LAN, it sends the packet to the central server which relays it onto its destination. In contrast, a peer to peer approach allows clients to create direct secure connections with each other. This approach has the potential to remove the central server as a potential bottleneck for network traffic. It also reduces or possibly eliminates the need for a centralized management and authentication system.

3.7.7 The use of a P2P-VPN allows for a virtual network topology to be defined in software regardless of the underlying physical topology. Therefore, if operating a P2P-VPN across a traditional physical network or utilizing for virtual network connectivity in the same box, the operating systems are unaware and do not have to be modified. Some P2P-VPN software packages such as N2N [40] allow for VPNs at Layer 2. A Layer 2 VPN transports and secures entire network frames. This is in contrast to a Layer 3 VPN such as IPOP [54] that works at the IP transport layer of the TCP/IP stack. The benefit of using a Layer 2 solution is that most hypervisors allow some type of bridging mechanism from the virtual Ethernet adapter inside the virtual machine to an adapter on the host. This additional level of network layer abstraction allows a software defined network topology of virtual machines to be connected together regardless of hypervisor or physical location of the host. This feature provides a great deal of flexibility when building training ranges in terms of scale of the range as well as the physical location of the end nodes (i.e., the students).

3.7.8 While the initial goal of this concept is to provide a stand-alone platform to train and educate cyber technologies, it may also provide long term additional capabilities by allowing separated stand-alone platform virtual networks to connect. This additional capability would allow two students to build independent networks and then connect together to participate in localized red/blue exercises. Ultimately, incorporating the ability to provide independent stand-alone virtual network platforms with the ability to connect independent virtual networks using VDE or P2P-VPNs can be used to significantly increase the effective network footprint used in the large-scale ranges previously discussed.

3.7.9 In summary, there is a wide range of existing projects that utilize virtualization techniques to provide an education platform for cyber security education. The designers of these platforms tend to choose an individual virtualization technique based on the requirements and performance requirements their platform must support. It is possible, however, to combine some types of virtualization on the same host. Doing so allows a cyber training platform to take advantage of the benefits of high performance virtualization techniques such as operating system virtualization, while still providing the same feature set as the more resource intensive full virtualization. Another need in a training platform is the ability to connect virtual machines. Providing this capability through peer to peer virtual private networks enables a layer of flexibility in how virtual machines connect to each other, independent of the underlying physical topology. This combination of host and network virtualization provides the conceptual building blocks useful in developing a platform for cyber education and training.

IV. Methodology

4.1. Overview

4.1.1 In order to validate the role of hybrid virtualization and peer to peer virtual private networks in building a platform for cyber education and training, it is important to understand the performance characteristics of each technology. Each category of virtualization has developed out of the desire to achieve some balance of features, capabilities and performance. This chapter provides a methodology that evaluates quantitatively the performance characteristics of the design discussed in Chapter III. The first section presents the methods to compare the traditional methods of virtualization to a novel approach that combines two types of virtualization simultaneously on the host platform. This includes important design criteria that are critical for properly evaluating the performance characteristics of host based virtualization. The final section presents a method that evaluates the performance characteristics of a peer to peer virtual private networking technology relative to direct network connections.

4.1.2. Methodology Components

4.1.1 The remainder of this chapter partitions the problem space into two parts. The first part presents a methodology to analyze the role of virtualization at the host level. Specifically it compares three classical virtualization techniques with the hybrid approach. The first part of host based virtualization section explains the special design consideration needed in measuring the performance of different hypervisors, each of which is designed to support a unique environment. The final part presents the experimental setup that creates similar environments for each type of virtualization

platform and then performs a variety of benchmarks in each platform. This provides a common operating environment to determine how each virtualization platform performs comparatively.

4.1.2 The final portion of this chapter presents a methodology to analyze the performance characteristics involved with virtualization at the network level. This portion examines the use of a peer to peer virtual private networking technology as a method for providing network virtualization of virtual machines. Two benchmarks types provide both latency and throughput performance metrics for each virtualization platform. The experiments perform these benchmarks both with and without the network virtualization in place in order to determine the amount of overhead compared to direct physical connections.

4.2. Host Based Virtualization Experiments

4.2.1. Performance Analysis Experiments

4.2.1.1 A hybrid virtualization technique is proposed as a possible solution to providing a more efficient virtualization platform for situations involving several virtual machines. One of the primary measures of performance of a computer system is the time required for that system to perform a fixed set of computational tasks. The purpose of this experimental setup is to create an environment for each virtualization platform where a standard set of benchmarks execute in order to determine the performance profile of that platform.

4.2.2. Design Considerations

4.2.2.1 Measuring the performance of a computer system can take on many different forms depending on the application. Virtualization brings into account its own unique set of design challenges when it comes to performance analysis. One aspect of this research that is especially challenging is designing a workload that will execute on each type of virtualization technology with the minimal amount of changes. It is important that the differences in the performance characteristics for each technology be representative of the performance capability of the hypervisor and not interference from an outside source. For example, if a test measures the time taken to compile a program from source, care should be taken so that the compiler and the libraries the benchmark depends on are the same from hypervisor to hypervisor.

4.2.2.2 One major difficulty with maintaining consistency in the performance workload is the fact that some hypervisors require extensive modification of the host kernel. This means that the host operating system itself must be customized for the hypervisor. The host kernel can have a significant impact on the performance of a system, since it is the core engine that drives everything from disk access to process and thread scheduling. In addition to modifying the host kernel, hypervisors such as OpenVZ and Xen place specific requirements on the guest operating environment. In the case of OpenVZ, the guest kernel is the same as the host kernel. Therefore the guest kernel is also intrinsically tied to the OpenVZ hypervisor. In the case of Xen, the privileged host Dom0 kernel is often different that the unprivileged guest DomU kernel (though it is possible to have a kernel that can function as both). Either way, both the guest and the

host kernels also have to have modifications that are Xen specific. Furthermore, these modifications cannot be applied to an arbitrary Linux kernel. The patches and code provided by Xen only work with kernels provided by the Xen project.

4.2.2.3 As hypervisors (especially those that have been designed to extract higher performance in their virtualization paradigm) have competing and often conflicting requirements, the difficulty of maintaining a consistent environment for a workload becomes more apparent. An important question becomes what a viable approach should be in order to minimize the changes that must occur from hypervisor to hypervisor. The approach taken in this research is to use the Linux distribution CentOS as a base operating system. CentOS is the freely distributed repackaging of the commercial Red Hat Enterprise Linux (RHEL) distribution. CentOS distinguishes itself for this type of performance analysis in two ways. First, it is based on a very stable and widely accepted code base from Red Hat's flagship enterprise operating system. Secondly, it provides the widest and most stable integration of the hypervisors under consideration in this research. So although the hypervisors technologies require all the modifications listed above, a common distribution helps to minimize the effects of these. Each CentOS kernel is based on a heavily patched 2.6.18 Linux kernel, forked and maintained by Red Hat. Both the Xen and OpenVZ provide kernels that are based on the same Red Hat kernel that CentOS uses. At least with regards to the host, this common kernel code base helps to minimize possible interference from sources other than the overhead required by the hypervisor.

4.2.2.4 Another design consideration is the guest operating system. As a baseline, the same considerations that were made for the host operating apply to the guest

operating systems as well. In the case of OpenVZ, since CentOS is the host operating system, it by default also becomes the guest operating system since the kernels are shared between the two. In the case of Xen, CentOS provides an unprivileged DomU kernel to use. With any of the full virtualization techniques, since they support a large range of guest operating systems, running CentOS presents no particular challenge.

4.2.2.5 One virtualization technology that does present a challenge is the case of User Mode Linux. Although User Mode Linux has been rolled into the official source tree for Linux kernels 2.6 and above, CentOS does not support being compiled for the User Mode Linux architecture. It is possible to utilize a non-CentOS kernel on a CentOS file system, but this begins to disrupt the consistency levels between workloads on each hypervisor. Since the CentOS kernel is heavily patched and customized for an enterprise environment, it can be difficult to compile a User Mode Linux that will match the CentOS kernel feature for feature. For this reason, User Mode Linux is not included in this study.

4.2.2.6 Another design consideration is the load that will be applied to each hypervisor. In the case of this study, the load is represented by virtual machines. So as is the case for the host operating system, the guest operating systems must be as close as possible to each other with respect to each hypervisor. The challenge here is that each virtualization technique presents a unique perspective to the guest operating system with respect to what hardware is available or even what kernel the guest should run. For example, both paravirtualization and full virtualization provide abstract hardware devices to the guest operating systems. Both systems also have virtual hard drives that are

typically represented by a file on the host operating system. The virtual machine interacts with this file as if it was a physical hard drive. Operating system virtualization does not fit this paradigm. Files for the guest machine exist directly on the host hard drive and access to the guest happens in a much more direct manner. This makes it difficult to present an identical load to each hypervisor. The changes applied to the load for each hypervisor should be minimal and should not be related to aspects of the benchmark itself (compiler and library versions, benchmarking code, etc.).

4.2.2.7 The final design consideration is that the benchmarking processes must be repeatable. This implies a certain level of automation that must be achieved so that the variance between samples is minimized. The automation must provide the means for each hypervisor to conduct the performance analysis tests with as few as possible hypervisor-specific changes as possible. This level of automation is achieved through the use of custom scripts that clone the base configurations, start the virtual machines and control the timing of the benchmarks.

4.2.2.8 In summary, the performance analysis of any hypervisors that require extensive and mutually exclusive modifications to the host or guest operating systems is going to face a challenge when it comes to eliminating the interference due to those modifications. This research attempts to minimize this interference by using the same CentOS distribution as the host and guest operating system and performing the benchmarks in a repeatable, automated fashion.

4.2.3. Benchmarks

4.2.3.1 In order to provide flexibility and stability in the benchmarks, the design uses a prepackaged suite of benchmarks called Phoronix Test Suite [55]. Phoronix Test Suite is a benchmarking suite written in PHP designed for automating the process of benchmarking Linux systems. It provides a command line interface to automatically download and install various commonly used benchmarks. It also automates the process of running multiple benchmarks in a suite. The cross platform nature of PHP makes it a natural fit to run both on the guest and host platforms.

4.2.3.2 All performance tests were carried out on hardware with the following specifications:

- Dell PowerEdge 860
- Intel Xeon 3060 Conroe 2.4GHz 4MB L2 with 1066 MHz bus and Intel VT-x virtualization
- 4 GB of RAM
- 2 x 80 GB hard drives
- 2 x Network Interface Cards

4.2.3.3 Separate full installs of the CentOS operating system were made to several partitions on the first 80 GB hard drive, one partition for each hypervisor under test and one clean install to use as a comparative baseline. A minimal CentOS install was made as a guest for each hypervisor. The Phoronix Test Suite along with the necessary supporting tools to execute the benchmarks was installed to both the guest and host systems.

4.2.3.4 Three benchmarks from the Phoronix Test Suite were selected to run across each of the platforms. These benchmarks represent real world tasks such as compiling software, encoding media files and compressing files. These tasks each provide a unique set of requirements on the subsystems of the virtualization platform including kernel and I/O performance. These benchmarks are also important for this type of testing because of the longer time duration of the test and the small variability of the results from run to run. The long time duration is necessary because the timer for the benchmark is not inside the virtual machines performing the test. This is because time inside a virtual machine can be inaccurate [56]. A longer duration test minimizes the effect of the overhead in the communication between a virtual machine and the host described below. Small variability between test runs ensures that a sample mean accurately reflects the performance of the hypervisor and helps to provide a more concrete comparison.

4.2.3.5 For each hypervisor, a base guest machine was created. This base image was then cloned to create six virtual machines. Each machine was configured to acquire a DHCP address from the host machine on boot. A custom script was placed in the systems startup files that initiated contact to the host. Another script on the host waits for guest machines to connect. When the scripts connect, they exchange information about which benchmark to perform and any other relevant data. At this point the virtual machine script waits for the signal to proceed with the benchmark. When the host is ready and all the virtual machines are connected, it begins an instance of the benchmark on the host. The script then immediately gives commands to each virtual machine to begin the benchmark.

The host script times how long it takes for the entire process to complete. This time is the metric used in the results section of Chapter V.

4.3. Network Based Virtualization Experiments

4.3.1. Virtualization of the Network Layer

4.3.1.1 Depending on the needs of the training environment, providing a mechanism for decentralizing and virtualizing the network layer may provide unique benefits in terms of the complexity of virtual network topologies. The decoupling of virtual network topology from the underlying physical network topology allows for a variety of training environments. One major advantage of this approach is the ability to easily provide distance learning capability to a training platform. Students would be able to connect virtual machines on their host machines from anywhere on the public Internet and connect to virtual training environments as if they were in the classroom.

4.3.1.2 Another benefit of virtualizing the network environment is that it provides a mechanism to define virtual network topologies programmatically. This can be a very useful tool when students are learning to create network topologies and understand the various concepts involved with large scale computer networks. Network virtualization allows students to create their own network topologies with their host node and seamlessly integrate their network with other networks created by other students and the instructor. The result is a very realistic environment that has been produced entirely at the software level and does not require complex and expensive physical equipment and resources.

4.3.2. Experimental Setup

4.3.2.1 The purpose of the network virtualization experimental setup is to evaluate the performance characteristics of network virtualization techniques compared to direct physical connections. The experiments focus on the performance metrics of network latency and network bandwidth. The performance measurements in these experiments use the same server cluster described in 4.2.3.2 connected via a Cisco 2600 10/100 switch. For this discussion the server initiating the connection is referred to as **T** and the receiving computer as **R**.

4.3.3. Benchmarks

4.3.3.1 The ping utility provides the network latency measurement. The ping utility on **T** sends a small ICMP Echo Request packet to **R** and begins a timer. When **R** receives the ICMP Echo Request it sends an ICMP Echo Reply packet. When **T** receives the ICMP Echo Reply, it stops the timer and reports the total amount of time elapse since sending the ICMP Echo Request. For this experiment, **T** first clears its ARP cache and sends a ping to **R**. This first ping with a clean ARP cache forces **T** to send out an ARP request first in order to determine the MAC address of **R**. This measures the amount of initial connection latency. **T** then sends a second ping now that it has the MAC address of **T**. This second ping measures the resolved connection latency.

4.3.3.2 The network benchmarking utility TTCP provides the bandwidth measurement. TTCP is capable of sending varying sized loads over the network and acts as both the transmitter and receiver of network packets. TTCP provides timing and bandwidth measurements on both the transmitting and receiving nodes. TTCP creates a

fixed size memory buffer on **T** and continuously sends the contents of this buffer a configurable number of times. The advantage of using a memory buffer as the source compared to a file is that it focuses the bandwidth measurement on the performance of the network rather than the local file I/O performance of the host. The experiment sets up a TTCP transmitter on **T** and a TTCP receiver on **R**. The experiment uses the default configuration options for TTCP except for varying the total number of bytes sent through the network. The experiment varies this factor by 16MB, 32MB, 64MB and 128MB. TTCP measures the amount of time to send the specified number of bytes through the network, indicating the amount of bandwidth available between the two nodes.

4.3.3.3 The experiments use latency and bandwidth tests in various configurations in order to provide a comparison between physical and virtualized network connections. The baseline is two servers that are physically connected and not running any virtualization software. The tests are repeated using VirtualBox, Xen and OpenVZ virtual machines as **T**. These virtual machines use virtualized network connections to connect to a physical machine **R** that also uses a virtualized network connection. The P2P VPN solution N2N is used to provide the virtual network connections. This provides data to compare against the baseline configuration described above. This set of experiments does not represent a full factorial design. Specifically, this set of experiments does not include performance data on virtual machines with physical connections. The reason for limiting the scope of the experiments to virtual machines with virtualized network connections is that this represents the target configuration of the system described in section 3.6. Also hybrid virtualization is not used in this set of experiments. Since the network connections

are from a single virtual machine to a physical machine, the hybrid virtualization technique would use either VirtualBox or OpenVZ to perform this experiment. Therefore the results would be the same as the results for either of those hypervisors individually.

V. Results

5.1. Host Based Virtualization Results

5.1.1 The tests were performed by running the designated benchmark simultaneously inside each running virtual machine as well as on the host system. The total time to complete the benchmark was recorded on the host. The selected benchmarks measured the time needed to configure and compile an Apache web server, encode a large audio file to MP3 format and compress a large file using the 7zip format. For the initial set of experiments, three samples for each data point were collected. While this does not represent enough samples to perform a rigorous statistical analysis it does illustrate some general behavior for each benchmark. The largest confidence intervals were ±9.6 sec, ±4.1 sec and ±2.3 sec for the Compile Apache, 7zip Compression and Encode MP3 benchmarks respectively.

5.1.2 These results show that performance between hypervisor techniques varies for each benchmark. Figures 12, 13 and 14 present the results from each of the benchmarks. Time is used as the metric of performance where lower times represent higher performance. The method used to collect these samples is described in 4.2.3.5. In the case of the hybrid approach, half of the virtual machines use the VirtualBox hypervisor and half use the OpenVZ hypervisor.

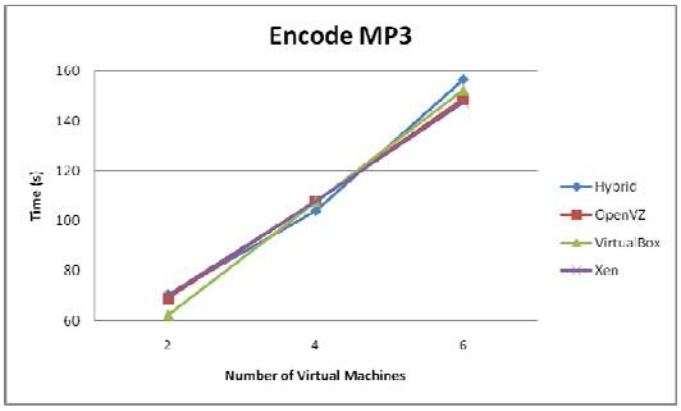

Figure 10: Results of the Encode MP3 benchmark

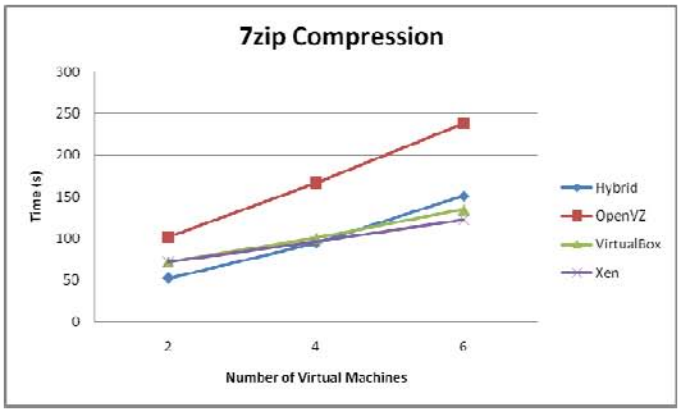

Figure 11: Results of the 7zip Compression benchmark

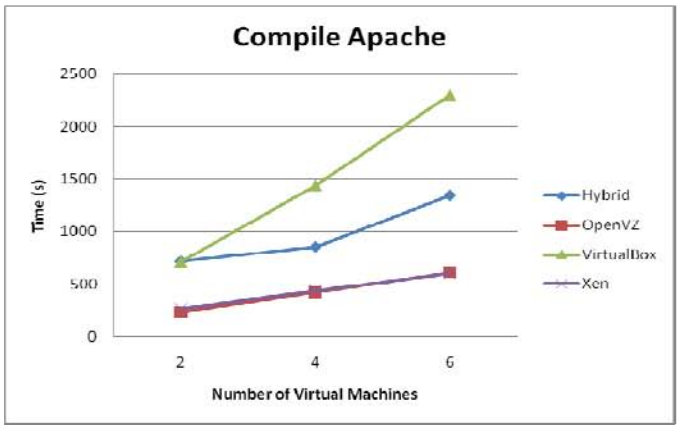

Figure 12: Results of the Compile Apache benchmark

5.1.3 The Encode MP3 benchmark shows consistent performance across each type of hypervisor. This illustrates a workload for which there is no distinct performance advantage for any one hypervisor. The 7zip Compression benchmark shows most of the hypervisors in a similar performance category with the possible exception of OpenVZ. OpenVZ was expected to outperform the other hypervisors due its low overhead. The 7zip compression benchmark illustrates an example where this might not be the case. The Compile Apache benchmark demonstrates the greatest performance differences amongst the hypervisors. The Compile Apache benchmark performs a significant amount of random file access during both the configuration and compilation phases. The virtual file I/O drivers in VirtualBox are most likely responsible for this slowdown. Issues regarding the performance of VirtualBox in compilation tasks have been reported on the VirtualBox forums [57]. VirtualBox performed significantly worse with Xen and OpenVZ comparing similarly to each other. The hybrid approach fits squarely in the middle.

5.1.4 Within the same benchmark, there appears to be little variation in performance from run to run on the same hypervisor. This implies that the data presents a good representation of how hypervisors perform on each particular task. However, since the relative performance of the hypervisors changes so drastically from task to task, it appears that there may not be a clear champion in terms of overall hypervisor performance. In other words, the results from these benchmarks are only representative of these tasks and do not appear to apply in a broader sense to each hypervisor.

5.1.5 Although it is not possible to draw general conclusions about each hypervisor with respect to its relative performance, the benchmarks do highlight some interesting advantages of the hybrid approach. The best example is found in the Compile Apache benchmark. Here the performance of the hybrid technique falls between full virtualization and operating system virtualization. In this case, the hybrid technique provides all the capability advantages of full virtualization, but is also taking advantage of some performance benefits of operating system virtualization. This performance gain is dependent on both the mix of virtual machines on the hybrid system as well as the workload of each virtual machine.

5.2. Network Based Virtualization Results

5.2.1. Introductory Clarifications

5.2.1.1 N2N uses software network devices called TAP/TUN devices to provide its virtual networking capabilities [40]. Although OpenVZ claims to support TAP/TUN devices inside virtual machines and provides steps to enable this environment [58], these steps did not work with setup described in the methodology in Chapter IV. Specifically, N2N was unable to establish a TAP device inside an OpenVZ virtual machine. One possible cause for this is due to the difference between persistent and non-persistent TAP devices. N2N relies on the creation of a persistent TAP device, but OpenVZ appears to only support non-persistent or transient TAP devices. The result of this is that data could not be collected for the OpenVZ virtual machine configuration. This illustrates a potential drawback to the OpenVZ approach to virtualization.

5.2.1.2 The following notation is used throughout this section to describe the experimental configurations:

- **Physical**/**N2N**: Describes the presence or absence of network virtualization respectively
- **Host**/**VM**: Describes if the tests were run from the host or inside a virtual machine
- **Novirt**/**Xen**/**OpenVZ**/**Vbox**: Describes the hypervisor used to conduct the experiment with Novirt representing no virtualization installed and Vbox representing VirtualBox.
- **Ping1**/**Ping2**/**16mb**/**32mb**/**64mb**/**128mb**: Describes the result of the first ping, second ping and various loads sent by the TTCP benchmark respectively
- **T/R:** Describes if the data is from the transmitter or receiver respectively in the TTCP benchmarks

5.2.2. Ping1 Latency Results

5.2.2.1 The first ping experiment is the initial ping that is sent from the transmitting computer **S** to the receiving computer **R**. Since the ARP cache line for the target IP address is cleared before the ping is sent, this first ping represents the amount of connection setup overhead. The results of each ping were written to a file and the experiment collected 30 samples per configuration. When the data was collected for analysis, it became apparent that the data was not normally distributed. Figure 13 gives the histograms for each configuration in the experiment.

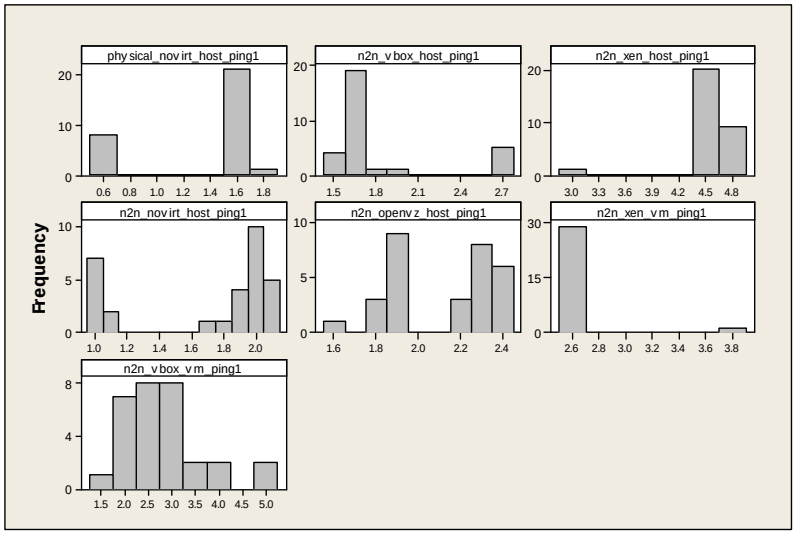

Figure 13: Histograms for each Ping1 experiment configuration

5.2.2.2 In the baseline sample, there appears to be two scenarios affecting the sample mean. Performing an Anderson-Darling Normality Test in the statistical package Minitab produces the graph shown in Figure 14. This shows that there are clearly two means present during this experiment. The first mean is based on conditions that leads to a sample mean value near 0.6 ms. The second condition leads to a sample mean of nearly 1.6 ms. There are 7 samples that fall in the 0.6 ms grouping compared to the remaining 23 samples in the 1.6 ms grouping.

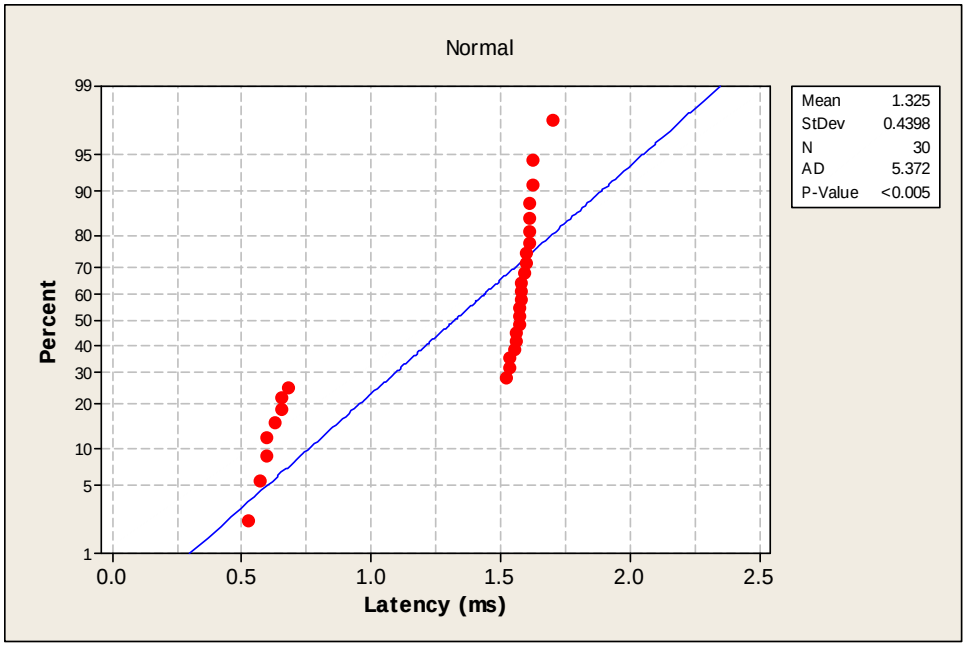

Figure 14: Normal Distribution Probability Plot for Ping1 experiment on a host with no host or network virtualization

5.2.2.3 The reason for the faster latency sample mean is that the ARP cache line for the destination machine is filled after the cache line is cleared by the script but before the first ping is sent out on the network. One possibility for this behavior is other networking software also running on the host that is able to force an ARP resolution to happen in between the time when the ARP cache line is cleared and when the ping utility forces a new ARP request to be sent. This would result in a reduced round trip time since the overhead of the initial connection would have already been met. If this is the case, the smaller cluster is actually samples from a configuration represented in the ping2 study. If we make this assumption and eliminate these samples from the sample population, the resulting normality probability plot looks like the graph presented in Figure 15.

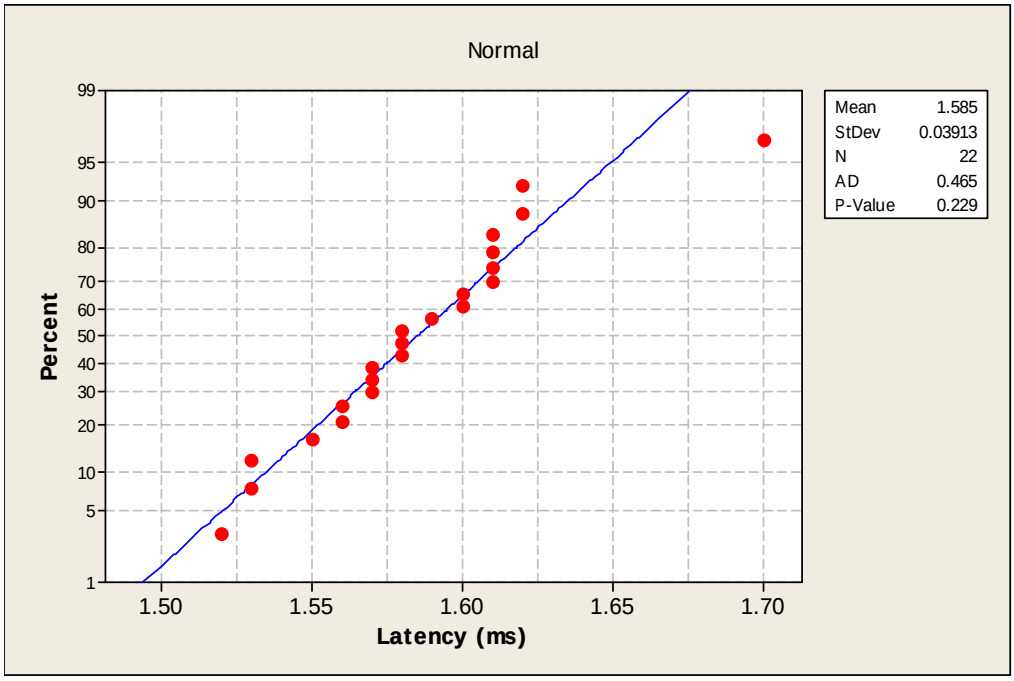

Figure 15: Normality distribution plot after removing lower cluster samples

5.2.2.4 The graph presented in Figure 15 demonstrates that when the higher latency sample cluster is examined individually, the results are a much closer match to a normal distribution with one high latency outlier. These types of high latency outliers may be the result of special case bandwidth consumption on the switch (due to a burst of broadcast packets for example) or a brief abnormal load on either the host or target machine. In order to perform an actual statistical analysis for this experiment, the effects of the low latency samples must be isolated at the experiment level. Due to the multiple possible causes involved, doing so falls outside the scope of this research document. Foregoing a formal statistical analysis, the raw results of the sampling is present in Table 2. The “Overhead” column is the result of dividing the sample mean of a particular configuration by the baseline sample mean.

Table 2: Results of Ping1 Experiment

Network	Hypervisor	Location	Latency (ms)	Overhead	Std Dev
physical	none	host	1.325	1.000	0.440
n2n	none	host	1.691	1.276	0.443
n2n	vbox	host	1.797	1.356	0.395
n2n	openvz	host	2.111	1.593	0.242
n2n	xen	vm	2.691	2.031	0.216
n2n	vbox	vm	2.807	2.118	0.882
n2n	xen	host	4.574	3.451	0.332

5.2.3. Ping2 Latency Results

5.2.3.1 The results from the Ping2 set of experiments conform much closer to a normal distribution. This is because it is easier to isolate the low latency case compared to capturing multiple samples of the initial connection setup latency. Figure 16 shows an example normal distribution probability plot for the same configuration presented in the Ping1 examples. There is one outlier and the distribution has small amount of bimodal skew as indicated by the ‘S’ shaped bends in the plot. When compared to the Ping1 results shown in Figure 14, the data is much closer to a normal distribution and is suitable for more thorough statistical analysis.

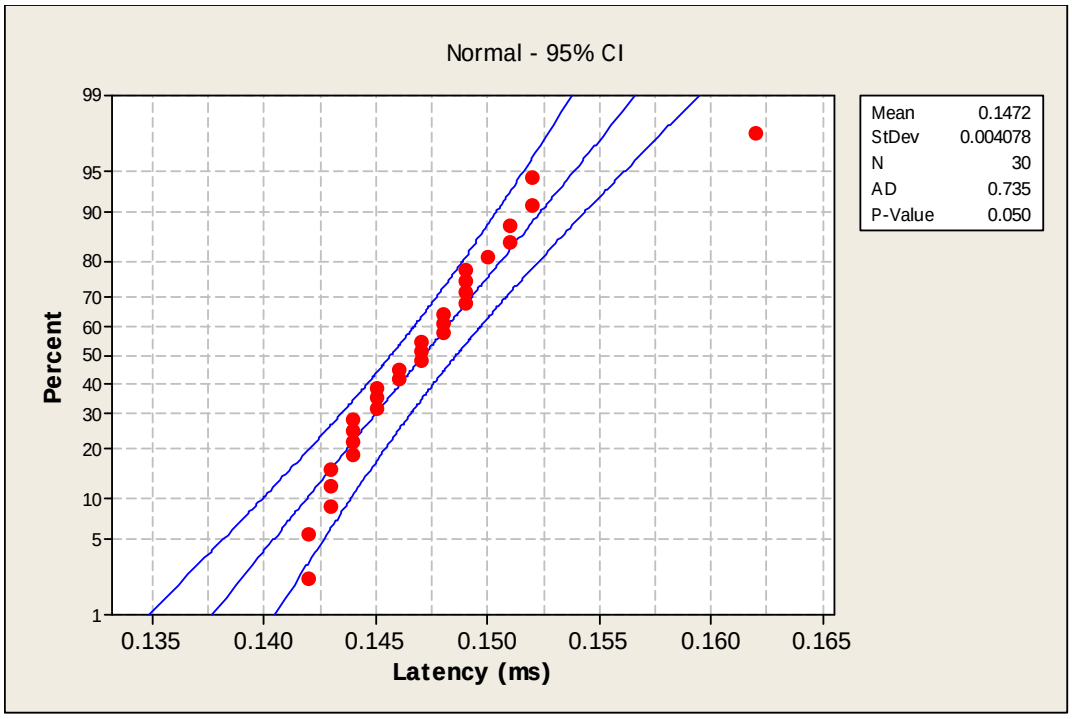

Figure 16: Normal Distribution Probability Plot for Ping2 experiment on a host with no host or network virtualization

5.2.3.2 The results for the Ping2 experiment are presented in Table 3 and the box plot comparing the results is shown in Figure 17. The "Confidence" column in Table 3 indicates the calculated confidence interval for the sample mean. The results fall into rough three groups. On the low side, the physical connection has an expectedly low latency relative to the network virtualization options. On the high side, The VirtualBox virtual machine shows a statistically significant higher latency than all the others. Three are also a significant number of outliers associated with the VirtualBox virtual machine. Although small variations appear amongst the other configurations, they are still relatively close to each other in terms of latency.

Table 3: Results for Ping2 Experiment

Network	Hypervisor	Location	Latency (ms)	Overhead	Std Dev	95% Confidence
physical	novirt	host	0.147	1.000	0.004	±0.001
n2n	openvz	host	0.520	3.536	0.015	±0.005
n2n	xen	vm	0.482	3.276	0.015	±0.005
n2n	vbox	host	0.407	2.765	0.015	±0.005
n2n	novirt	host	0.412	2.802	0.015	±0.005
n2n	xen	host	0.449	3.051	0.017	±0.006
n2n	vbox	vm	0.735	4.993	0.214	±0.076

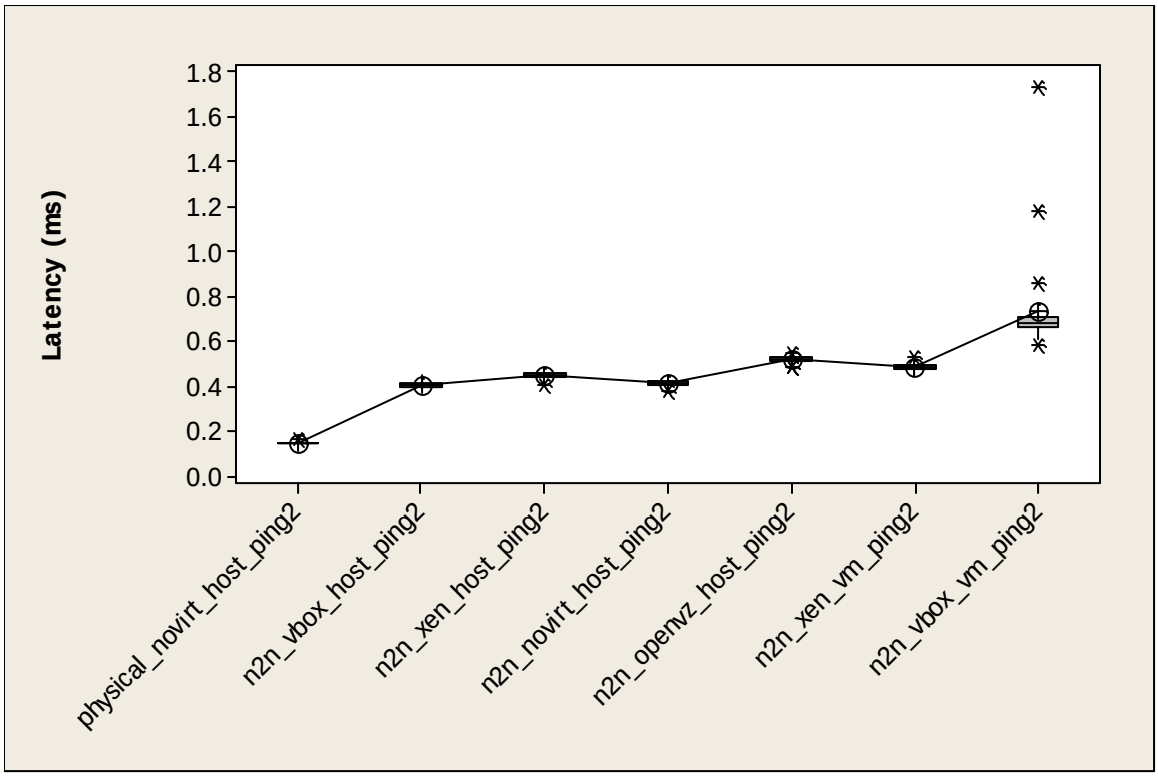

Figure 17: Box plot for the various Ping2 configurations

5.2.3.3 The Ping2 set of data provides the most insight into the effect of the network virtualization on the latency of the network connection. There appears to be about a threefold increase in latency for most of the configurations relative to a direct physical connection with no host virtualization. The choice of hypervisor has a relatively

small effect on the latency with the exception of VirtualBox which results in statistically significantly higher latencies and much wider range of latency values.

5.2.4. TTCP Bandwidth Results

5.2.4.1 Relative to the latency experiments presented above, the bandwidth experiments introduce a significant increase in the number of configurations. The experiment was repeated for each configuration presented in the latency experiments with the addition of four different levels representing the different sized loads. The latency experiments also indicate that there is a possibility for non-normal distributions in the data. So the first step in sifting through the large amount data from the experiments is to focus on the results that appear to represent normal distributions. Also due to the results in the latency experiments, a reasonable assumption is that the difference in performance between physical and virtualized connections represents the bulk of what differences might exist.

5.2.4.2 Using these assumptions as a guide, the following sequence of figures present graphs for the physical and virtualized network connections on systems with no host based virtualization installed. The probability plots in these graphs illustrate how closely the sampled data represents a normally distributed data set. The curves represent the fit to a normal distribution at a 95% confidence interval. A linear trend in the data that falls within the 95% confidence interval boundaries indicates a good fit to a normal distribution.

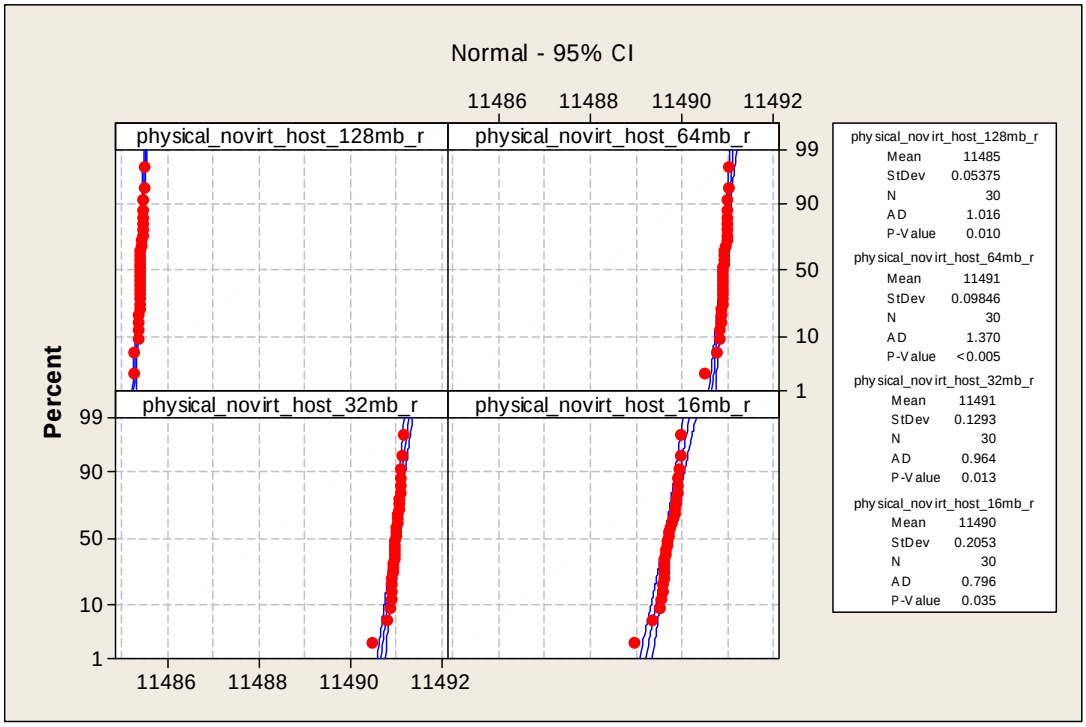

Figure 18: Normal distribution probability plot for TTCP receiving on physical connection

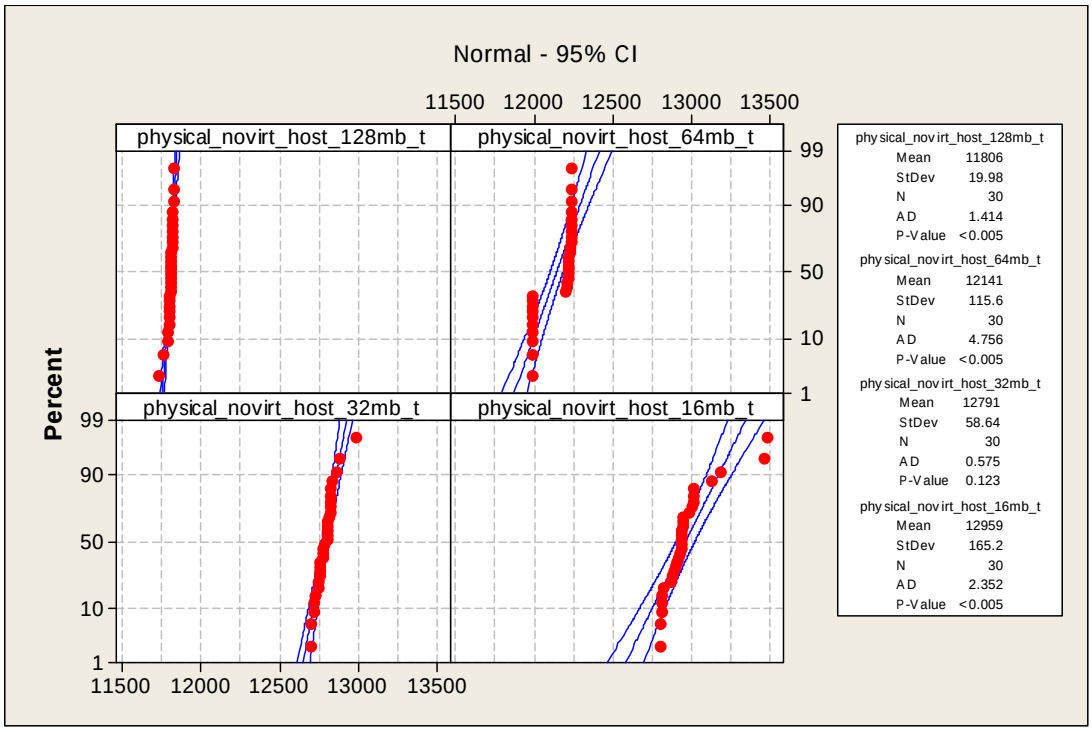

Figure 19: Normal distribution probability plot for TTCP transmitting on physical connection

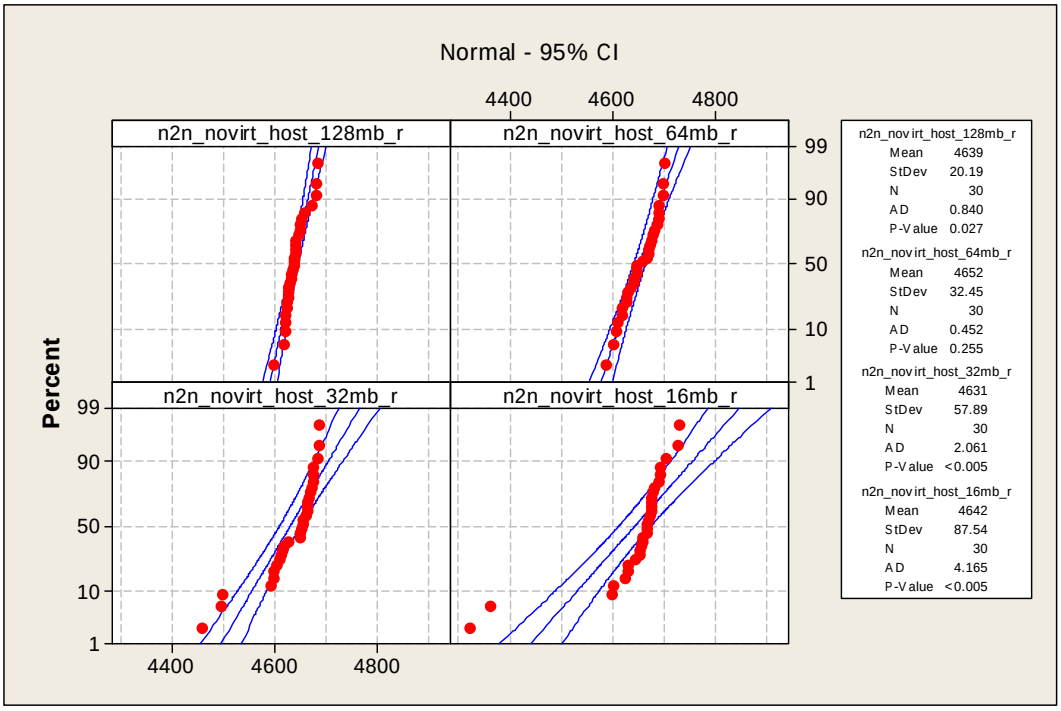

Figure 20: Normal distribution probability plot for TTCP receiving on virtualized connection

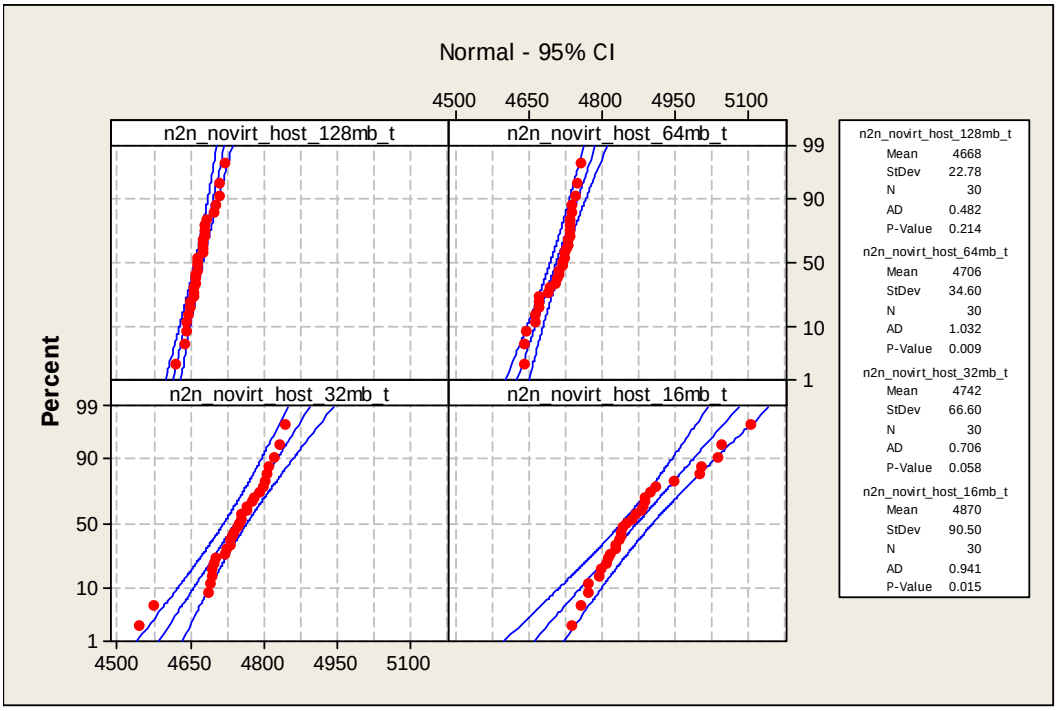

Figure 21: Normal distribution probability plot for TTCP transmitting on virtual connection

5.2.4.3 The plots show various degrees of fit to a normal distribution. Many of the 16mb profiles show the greatest degree of deviation from a normal distribution. This is likely due to the fact that the short amount of time required to transmit the 16mb load over the network allows uncontrolled factors in the experiment to affect the amount of bandwidth available on the network at the time of the experiment. The probability plots also show that the 128mb load is the configuration that stays closest to the expected behavior of a normal distribution. This behavior should be expected with a larger network load since the disruptions that become noticeable in the shorter bursts get smoothed out as the transfer takes longer in time.

5.2.4.4 Given the results from the probability plots, the 128mb appears to be the most stable distribution of data points from the four load sizes. Figure 22 shows the box plot of the TTCP transmit bandwidth results for the 128mb load.

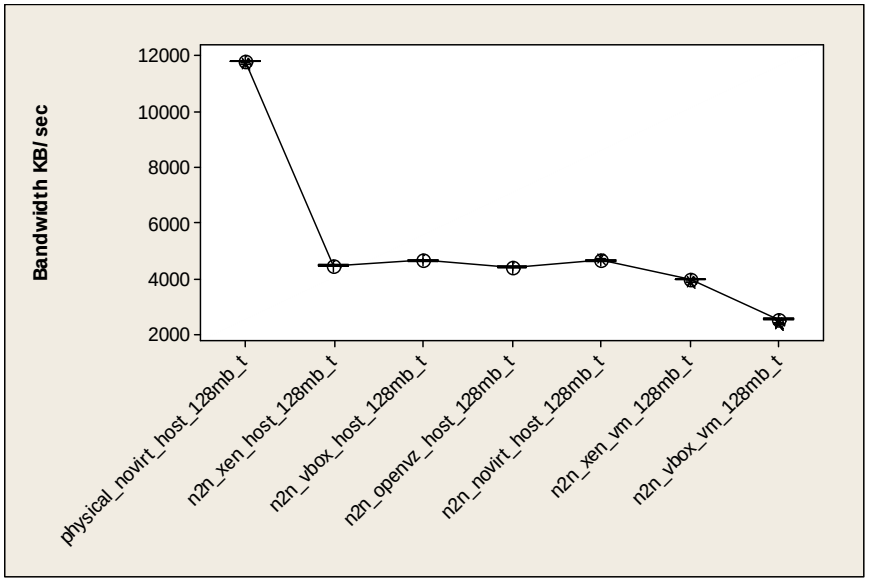

Figure 22: Box plot of the TTCP transmitters in the 128mb transmission load

5.2.4.5 The box plot describes a similar situation as the results from the latency experiments. There is a large, statistically significant difference between the physical connection and the virtualized connections. Amongst the host based virtualized connections, some differences exist but are relatively minor. Both the Xen and VirtualBox show measureable and statistically significant reduction in bandwidth, with VirtualBox showing the highest reduction in performance. The results for the receiving side of the same TTCP transmission are shown in Figure 23. These results are almost identical to the box plot for the transmitting node shown in Figure 22.

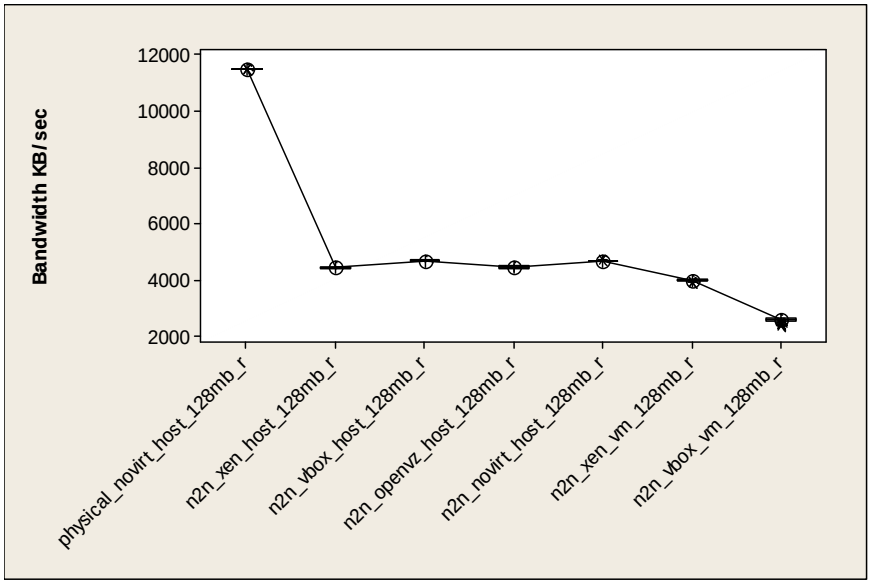

Figure 23: Box plot of the TTCP receivers in the 128mb transmission load

Table 4: Results for the 128mb TTCP transmit bandwidth

Network	Hypervisor	Location	KB/sec	Overhead	Std Dev	95% Confidence
n2n	vbox	vm	2551.549	4.627	71.030	±25.417
n2n	xen	vm	3981.741	2.965	21.017	±7.521
n2n	openvz	host	4429.964	2.665	57.669	±20.636
n2n	xen	host	4482.822	2.634	22.005	±7.874
n2n	novirt	host	4668.067	2.529	22.785	±8.153
n2n	vbox	host	4678.980	2.523	36.049	±12.900
physical	novirt	host	11806.356	1.000	19.983	±7.151

Table 5: Results for the 128mb TTCP receive bandwidth

Network	Hypervisor	Location	KB/sec	Overhead	Std Dev	95% Confidence
n2n	vbox	vm	2546.520	4.510	69.183	±24.756
n2n	xen	vm	3965.723	2.896	20.360	±7.286
n2n	xen	host	4408.448	2.605	18.826	±6.737
n2n	openvz	host	4425.594	2.595	57.467	±20.564
n2n	novirt	host	4639.243	2.476	20.191	±7.225
n2n	vbox	host	4649.025	2.470	32.881	±11.766
physical	novirt	host	11485.411	1.000	0.054	±0.019

5.2.4.6 Table 4 and Table 5 give the numerical results for the TTCP 128mb experiment. The “Overhead” column is calculated by dividing the sample mean for that particular configuration by the baseline physical connection with no host virtualization.

5.2.4.7 There are many different techniques for providing desktop virtualization. Each technique comes with its own unique set of advantages, disadvantages and

performance profiles. Unfortunately there does not appear to be a silver bullet when it comes to virtualization. In the small test workloads presented here, the performance profiles have been both very similar and very different depending on the specific benchmark under test. The benchmark that best illustrates the use of a hybrid approach is the Compile Apache benchmark. Here the hybrid approach improves the performance of the full virtualization performance. Although the performance is not quite as good as either paravirtualization or operating system virtualization, the hybrid approach provides additional capabilities that these two techniques cannot provide. The primary capability is the hybrid's ability to virtualize arbitrary guest operating systems. Overall, the hybrid approach appears to be a success candidate for use as the primary virtualization technique for a cyber warfare training simulation environment. It has the ability to support a wide variety of operating systems and performs equal to or better that full virtualization alone depending on the workload.

5.2.4.8 Regarding network virtualization, there is an expected amount of overhead associated with this method. This is due to the additional layer in the network stack that must encrypt and package inbound and outbound packets so that they will travel in the virtual network. Experimental results show that in a local area network, this overhead results in approximately two to four times the performance loss relative to direct physical connections. The advantage of virtualized networks is that they allow for greater flexibility in defining network topologies that are independent of the underlying physical network topology.

VI. Conclusions and Recommendations

This chapter provides a summary to the research presented in this document as well as the direction for future research in this area. The chapter is divided into two sections. The first section summarizes the three main areas of research presented in this document, the results of the experiments that were conducted as well as the implications of those results. The final section outlines potential areas for future research in this area.

6.1. Conclusions

6.1.1 This paper covers three related areas of research regarding virtualization in education. The first area describes some of the ideas behind creating a platform for conducting cyber warfare education and training. Virtualization is a key component in creating realistic cyber battlefields that allows Airmen to acquire hands on training in this increasingly important battle space. Improving technology performance in this domain will lead to an improved capability to accurately model and simulate the cyber domain. This is a necessary step forward for properly conducting education, training and operations in cyber.

6.1.2 A variety of techniques exist to provide this virtualization capability, but each technique comes with its own set of unique advantages and disadvantages with regards to capability and performance. Full virtualization provides the most flexibility but incurs performance overhead to provide the layer of abstraction. Paravirtualization is able to improve performance over full virtualization by providing more integration between the guest and host. This performance gain comes at the cost of requiring modifications to the guest operating system, limiting the systems that can take advantage. Operating

system or container virtualization provides the highest performance but also the closest integration between guest and host.

6.1.3 Finding a balance between these competing technologies depends heavily on the requirements of the system under consideration. This research proposes a method that combines full virtualization and operating system virtualization together on the same host. This combination is possible because although each requires non-trivial modifications to the host, these modifications are mutually compatible. This research proposes an architecture that is capable of supporting full and container based virtual machines on the same host. This platform leverages the operating system flexibility of full virtualization along with the performance benefits of operating system virtualization.

6.1.4 The second area of research presented in this data is the set of experiments to determine the performance profile of the different approaches to virtualization, including the hybrid approach described above. Experimental data demonstrates that actual performance of these hypervisors is not always dramatically different. The performance characteristics depend on the task at hand. Two of the three benchmarks indicate no significant performance differences. One benchmark based on the time required to configure and compile the Apache web server did show a significant difference. In this case, the performance profile of the hybrid approach fell between full and operating system virtualization as expected. This shows that hybrid virtualization is capable of higher performance that full virtualization alone dependant on the workload as well as distribution between full and operating system virtual machines.

6.1.5 The final area of this research deals with the concept of network virtualization. Network virtualization is an important component of the cyber warfare education and training platform described in this research. Network virtualization allows for increased flexibility in the arrangement of virtual machines of varying hypervisor types. With network virtualization, the topology of a virtual network can be spread over a wide area network such as the public Internet. This topology can be software defined and independent of the underlying physical topology. This creates an environment where students have increased flexibility over the creation and connections of their virtual networks.

6.1.6 As with host based virtualization, this flexibility does not come for free. There is an overhead associated with the virtualization of the network layer. Additional software drivers must encrypt and repackage inbound and outbound packets destined for the virtual network. This research presents a set of experiments designed to characterize this performance overhead relative to direct connections. The experimental data indicates there is anywhere from a two to four times reduction in performance with respect to both latency and bandwidth when the network is virtualized over a direct physical connection.

6.2. Recommendations For Future Work

6.2.1 Although the hybrid approach shows some promise in the Apache benchmark, it is not without its drawbacks. Two of three benchmarks indicate there is not a significant amount of performance difference between various tasks. Although the two systems selected for this research are capable of working together on the same host, the experience is not entirely stable. Some issues arose during the experiment phase

regarding network connectivity. The biggest problem was the lack of support for persistent TAP/TUN drivers inside OpenVZ containers. This prevented a test of the N2N peer to peer virtual private networking software.

6.2.2 The recommendation of this research is that additional effort be applied to methods of integrating peer to peer virtual private networking solutions to a cyber warfare education and training platform. Given that performance of the different hypervisors depends heavily on the task at hand, those that use virtualization for education have most likely found a virtualization technique that suites their purposes. A direction for future work involves finding flexible and efficient ways to connect the resources of different academic institutions, regardless of the choice of underlying hypervisor, in a way that allows for the easy creation of wide area cyber warfare arenas. Other possible research areas include methods to increase the efficiency of current hypervisor methods, especially with regards to memory, disk and network usage.

6.2.3 As the Air Force pushes forward to fight in the Cyber domain, it is critical that Airmen have environments that allow for realistic training. Cyber is unique in the fact the tools best suited to model the domain are in the Cyber domain itself. Virtualization can create computer environment that in some ways can be nearly indistinguishable from computer systems running on traditional physical hosts. Research that can effective leverage this type of technology to create realistic cyber battlefields will help the United States Air Force maintain the leading edge in a growing and increasing contested Cyber domain.

Appendix A: Concepts in x86 Virtualization

With the wide scale adoption of the Windows operating system and Intel x86 based line of processors (affectionately referred to as Wintel), the x86 platform has nearly saturated the desktop market and a large portion of the Information Technology industry at large. Over time, physical hardware continued to mature and develop. An effect known as Moore's law has predicted a doubling in hardware performance every 12 to 18 months. This exponential rate of performance growth has lead to hardware designs that are increasingly more capable. Virtualization is one attractive use of this availability of performance. The ability to abstract the underlying hardware at the software level in a manner that does not significantly reduce performance introduces a wide variety of useful applications. Running multiple instances of what appears to be independent computer systems provides unique capabilities anywhere from creating fault tolerant setups to investigating operating system design principles.

In order to understand the way in which virtualization occurs on the x86 platform, it is important to review some basic principles relating to how processors implement security privilege levels. A processor will typically have several privilege modes available to the software it executes. This is to provide hardware level separation of operating system code and user code. The operating system kernel will typically execute at the most privileged level, giving it unrestricted access to memory and other hardware devices available. The operating system will then run all the other programs and software it manages at a lower privilege level. When these lower privileged programs require access

to hardware devices such as the hard disk or the graphics card, the program executes a system call. A system call allows the program to transfer control to the operating system so that it can execute any privileged instructions the unprivileged code is unable to execute itself. When the operating system completes the requested service, it returns control back to the code that made the system call and execution continues.

The Intel architecture specifies four privilege levels called rings [22]. They are labeled Ring 0 through Ring 3 with the former being the most privileged level and the latter being the least privileged. Although there are four rings available, in practice only two ring levels are utilized by mainstream operating systems such as Windows and Linux. Typically the operating system kernel itself operates at Ring 0 and all user code (that is everything besides the operating system) runs at Ring 3. When user code wishes to perform some type of privileged instruction (like interact with the hardware), the user code will make a system call. On the x86 platform, this is generally done either with a software interrupt or through the SYSENTER/SYSCALL instructions. illustrates the typical execution flows using the traditional x86 security model.

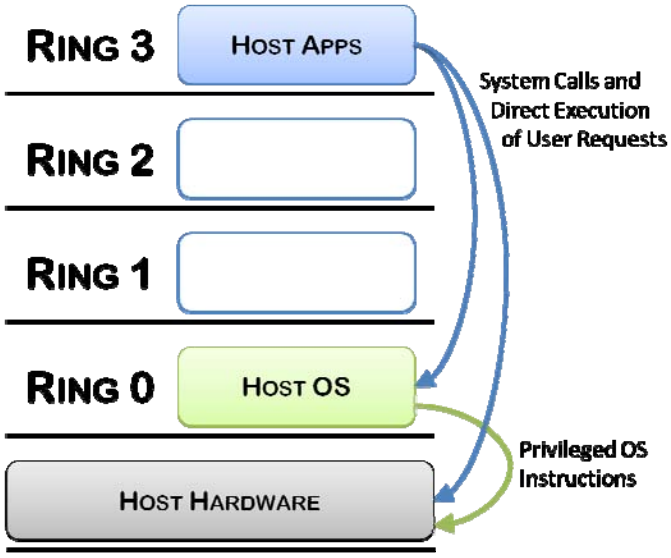

Figure A1: Traditional execution paths for the x86 security model

While the system call describes a mechanism where privilege is properly escalated, the CPU must account for unprivileged code attempting to execute privileged code on its own. Whenever a program attempts to execute a privileged instruction, the CPU uses the current privilege level to determine if the instruction should be allowed to execute. If the current privilege level is insufficient, the CPU generates a general protection fault. It is up to code at the next higher privilege ring to trap the fault and deal with the situation (terminate the offending program for example). The fault is trapped by the processor, which means the processor switches to the next highest privilege level that contains code that deals with the fault. Typically, since the operating system is the only other code present on the system and runs at Ring 0, the kernel is responsible for dealing with the fault. The following sections illustrate the importance that the ability to trap privileged instructions has in implementing virtualization.

The hardware requirements and characteristics necessary to properly support virtualization were laid out in a classic paper back in 1974 by Popek and Goldberg [14]. This paper lays out three fundamental characteristics that constitute a true hypervisor (referred to in the paper as a virtual machine monitor). First a hypervisor must create an environment wherein programs that execute in that environment must exhibit the same behavior as though they were executing on the real host. That is, from the perspective of the virtualized system it should not be able to determine that its instructions are being virtualized. Second, the efficiency of the virtualization must not substantially affect the speed. In other words, a statistically dominant set of the instructions must run at native speed and the overall overhead costs of virtualization must be insignificant. Finally, the hypervisor must be able to mediate all access of the virtual machine to physical resources such as CPU, memory, and I/O devices.

There have been significant hurdles to overcome to make virtualization on the x86 possible, let alone efficient. The x86 architecture was not originally designed with virtualization in mind. When virtualizing a platform such as the x86, code needs to be inserted into the flow of execution in such a way that from the operating system perspective the code acts like it is running physically on hardware when it is in fact being virtualized at a lower privilege level.

Hardware platforms that correctly implement the features described by Popek and Goldberg generally do so through a technique known as trap-and-emulate. This technique involves running a hypervisor at a more reduced privilege level than the operating system

running on the physical hardware but higher than what the level at which unprivileged code executes. In the case of the x86 platform, the real operating system runs at Ring 0, the real unprivileged code runs at Ring 3, the hypervisor typically runs at either Ring 1 or Ring 2 and the virtualized unprivileged code would also typically run at Ring 3. With this setup, the unvirtualized code continues to operate as before. Also the virtualized unprivileged code also executes as it would normally in an unvirtualized system.

The difference comes when unprivileged code executes a system call in order to execute privileged instructions. In the case of the virtualized system, the virtualized code running at Ring 3 would execute a system call. This results in a trap and the processor switches ring levels to the next privilege level that has code to handle the software protection fault that is generated. In the virtualized case, this would transfer control to the hypervisor running at either Ring 1 or Ring 2. This hypervisor creates an environment such that the virtualized operating system executes code as if it were in physical control of the hardware. This involves keeping track of virtual machine characteristics such as memory page tables, virtual hardware driver state and other internal data structures. At this point, the hypervisor virtualizes the request of the virtualized operating system and performs the action on a separate set of virtualized or real hardware components (such as memory, hard disks, and peripherals). Control may then return to the original point of execution where the system call originated.

The key in this process working properly is that all instructions must trap properly. In other words, when the virtualized operating system attempts to execute instructions that

can only be executed at Ring 0, the system must properly trap that call and execute code to handle the general protection fault. Unfortunately in the original design of the x86, there are several privileged instructions that instead of trapping when executed at a lower privilege level will instead just silently fail. This prevents virtualization software from properly intercepting the privileged instruction and virtualizing its effect. One example of this is the POPF instruction [23]. This is a privileged instruction that needs to execute in Ring 0 as it is used to set and clear the interrupt-disable flag. However, when this code executes at any privilege level other than Ring 0, the CPU simple ignores the instruction instead of generating a general protection fault that can be trapped by a hypervisor. This makes it impossible for a hypervisor to properly intercept the instruction when the guest kernel executes it in its unprivileged state.

Appendix B: Complete Host Virtualization Results

Xen (Host Benchmark time in sec)					
VMs\Iteration	1	2	3	Average	Std Dev
0	63.98	63.49	63.57	63.68	0.21
1	108.33	108.23	107.40	107.99	0.42
2	165.78	164.70	165.56	165.35	0.47
4	277.00	275.27	272.24	274.84	1.97
Xen (Total Time in sec)					
VMs\Iteration	1	2	3	Average	Std Dev
0	136.40	131.60	131.50	133.17	2.29
1	187.80	183.50	189.60	186.97	2.56
2	263.40	264.90	265.00	264.43	0.73
4	435.80	432.90	433.40	434.03	1.27
VirtualBox (Host Benchmark time in sec)					
VMs\Iteration	1	2	3	Average	Std Dev
0	47.61	47.43	47.39	47.48	0.10
1	53.96	54.53	53.82	54.10	0.31
2	60.69	61.09	59.67	60.48	0.60
4	80.80	78.42	76.71	78.64	1.68
VirtualBox (Total Time in sec)					
VMs\Iteration	1	2	3	Average	Std Dev
0	96.90	92.60	92.50	94.00	2.05
1	645.40	646.40	645.10	645.63	0.56
2	697.80	702.20	703.00	701.00	2.29
4	1442.90	1441.90	1476.80	1453.87	16.22
OpenVZ (Host Benchmark time in sec)					
VMs\Iteration	1	2	3	Average	Std Dev
0	52.05	51.80	51.37	51.74	0.28
1	94.73	94.32	94.04	94.36	0.28
2	146.97	147.93	148.19	147.70	0.52
4	280.90	282.80	278.90	280.87	1.59
OpenVZ (Total Time in sec)					
VMs\Iteration	1	2	3	Average	Std Dev
0	106.90	102.60	103.00	104.17	1.94
1	151.90	151.60	151.20	151.57	0.29
2	229.10	229.90	231.10	230.03	0.82
4	414.00	416.50	414.00	414.83	1.18

Hybrid (Host Benchmark time in sec)					
VMs\Iteration	1	2	3	Average	Std Dev
0	51.45	51.55	51.68	51.56	0.09
1	N/A	N/A	N/A	N/A	N/A
2	89.35	87.08	85.06	87.16	1.75
4	125.01	121.15	120.00	122.05	2.14
Hybrid (Total Time in sec)					
VMs\Iteration	1	2	3	Average	Std Dev
0	104.50	101.00	101.80	102.43	1.50
1	N/A	N/A	N/A	N/A	N/A
2	717.70	713.50	716.80	716.00	1.81
4	850.20	855.40	856.40	854.00	2.72
Baseline (Host Benchmark time in sec)					
VMs\Iteration	1	2	3	Average	Std Dev
0	47.10	47.66	47.43	47.40	0.23
Baseline (Total Time in sec)					
VMs\Iteration	1	2	3	Average	Std Dev
0	93.90	91.10	91.30	92.10	1.28

Appendix C: Complete Network Virtualization Results

Net-work	Hyper-visor	Loc-ation	Bench-mark	Factor	Exper-iment	Sample Mean	Std Dev	95% CI
physical	novirt	host	ping	default	ping1	1.325	0.440	0.157
physical	novirt	host	ping	default	ping2	0.147	0.004	0.001
physical	novirt	host	ttcpr	128mb	bw	11485.411	0.054	0.019
physical	novirt	host	ttcpr	128mb	bytes	134217728	0.000	N/A
physical	novirt	host	ttcpr	128mb	sec	11.410	0.000	0.000
physical	novirt	host	ttcpr	32mb	bw	11490.991	0.129	0.046
physical	novirt	host	ttcpr	32mb	bytes	33554432	0.000	N/A
physical	novirt	host	ttcpr	32mb	sec	2.850	0.000	0.000
physical	novirt	host	ttcpr	64mb	bw	11490.892	0.098	0.035
physical	novirt	host	ttcpr	64mb	bytes	67108864	0.000	N/A
physical	novirt	host	ttcpr	64mb	sec	5.700	0.000	0.000
physical	novirt	host	ttcpr	default	bw	11489.686	0.205	0.073
physical	novirt	host	ttcpr	default	bytes	16777216	0.000	N/A
physical	novirt	host	ttcpr	default	sec	1.430	0.000	N/A
physical	novirt	host	ttcpt	128mb	bw	11806.356	19.983	7.151
physical	novirt	host	ttcpt	128mb	bytes	134217728	0.000	N/A
physical	novirt	host	ttcpt	128mb	sec	11.101	0.018	0.007
physical	novirt	host	ttcpt	32mb	bw	12791.481	58.636	20.982
physical	novirt	host	ttcpt	32mb	bytes	33554432	0.000	N/A
physical	novirt	host	ttcpt	32mb	sec	2.562	0.013	0.005
physical	novirt	host	ttcpt	64mb	bw	12140.932	115.585	41.361
physical	novirt	host	ttcpt	64mb	bytes	67108864	0.000	N/A
physical	novirt	host	ttcpt	64mb	sec	5.399	0.051	0.018
physical	novirt	host	ttcpt	default	bw	12959.378	165.197	59.114
physical	novirt	host	ttcpt	default	bytes	16777216	0.000	N/A
physical	novirt	host	ttcpt	default	sec	1.265	0.015	0.005
n2n	novirt	host	ping	default	ping1	1.691	0.443	0.159
n2n	novirt	host	ping	default	ping2	0.412	0.015	0.005
n2n	novirt	host	ttcpr	128mb	bw	4639.243	20.191	7.225
n2n	novirt	host	ttcpr	128mb	bytes	134217728	0.000	N/A
n2n	novirt	host	ttcpr	128mb	sec	28.253	0.123	0.044
n2n	novirt	host	ttcpr	32mb	bw	4630.728	57.893	20.716
n2n	novirt	host	ttcpr	32mb	bytes	33554432	0.000	N/A
n2n	novirt	host	ttcpr	32mb	sec	7.078	0.091	0.032
n2n	novirt	host	ttcpr	64mb	bw	4651.609	32.454	11.613
n2n	novirt	host	ttcpr	64mb	bytes	67108864	0.000	N/A
n2n	novirt	host	ttcpr	64mb	sec	14.090	0.098	0.035
n2n	novirt	host	ttcpr	default	bw	4642.175	87.542	31.326
n2n	novirt	host	ttcpr	default	bytes	16777216	0.000	N/A
n2n	novirt	host	ttcpr	default	sec	3.530	0.070	0.025
n2n	novirt	host	ttcpt	128mb	bw	4668.067	22.785	8.153
n2n	novirt	host	ttcpt	128mb	bytes	134217728	0.000	N/A

Net-work	Hyper-visor	Loc-ation	Bench-mark	Factor	Exper-iment	Sample Mean	Std Dev	95% CI
n2n	novirt	host	ttcpt	128mb	sec	28.079	0.136	0.049
n2n	novirt	host	ttcpt	32mb	bw	4741.557	66.604	23.834
n2n	novirt	host	ttcpt	32mb	bytes	33554432	0.000	N/A
n2n	novirt	host	ttcpt	32mb	sec	6.913	0.098	0.035
n2n	novirt	host	ttcpt	64mb	bw	4705.566	34.603	12.382
n2n	novirt	host	ttcpt	64mb	bytes	67108864	0.000	N/A
n2n	novirt	host	ttcpt	64mb	sec	13.927	0.103	0.037
n2n	novirt	host	ttcpt	default	bw	4870.247	90.500	32.384
n2n	novirt	host	ttcpt	default	bytes	16777216	0.000	N/A
n2n	novirt	host	ttcpt	default	sec	3.365	0.062	0.022
n2n	openvz	host	ping	default	ping1	2.111	0.242	0.086
n2n	openvz	host	ping	default	ping2	0.520	0.015	0.005
n2n	openvz	host	ttcpr	128MB	bw	4425.594	57.467	20.564
n2n	openvz	host	ttcpr	128MB	bytes	134217728	0.000	N/A
n2n	openvz	host	ttcpr	128MB	sec	29.621	0.386	0.138
n2n	openvz	host	ttcpr	32MB	bw	4433.238	131.829	47.173
n2n	openvz	host	ttcpr	32MB	bytes	33554432	0.000	N/A
n2n	openvz	host	ttcpr	32MB	sec	7.398	0.228	0.081
n2n	openvz	host	ttcpr	64MB	bw	4426.166	122.922	43.986
n2n	openvz	host	ttcpr	64MB	bytes	67108864	0.000	N/A
n2n	openvz	host	ttcpr	64MB	sec	14.818	0.427	0.153
n2n	openvz	host	ttcpr	default	bw	4486.614	137.559	49.224
n2n	openvz	host	ttcpr	default	bytes	16777216	0.000	N/A
n2n	openvz	host	ttcpr	default	sec	3.656	0.118	0.042
n2n	openvz	host	ttcpt	128MB	bw	4429.964	57.669	20.636
n2n	openvz	host	ttcpt	128MB	bytes	134217728	0.000	N/A
n2n	openvz	host	ttcpt	128MB	sec	29.593	0.386	0.138
n2n	openvz	host	ttcpt	32MB	bw	4449.708	132.627	47.459
n2n	openvz	host	ttcpt	32MB	bytes	33554432	0.000	N/A
n2n	openvz	host	ttcpt	32MB	sec	7.370	0.227	0.081
n2n	openvz	host	ttcpt	64MB	bw	4434.850	123.379	44.150
n2n	openvz	host	ttcpt	64MB	bytes	67108864	0.000	N/A
n2n	openvz	host	ttcpt	64MB	sec	14.788	0.427	0.153
n2n	openvz	host	ttcpt	default	bw	4524.928	140.772	50.374
n2n	openvz	host	ttcpt	default	bytes	16777216	0.000	N/A
n2n	openvz	host	ttcpt	default	sec	3.624	0.118	0.042
n2n	vbox	host	ping	default	ping1	1.797	0.395	0.141
n2n	vbox	host	ping	default	ping2	0.407	0.015	0.005
n2n	vbox	host	ttcpr	128mb	bw	4649.025	32.881	11.766
n2n	vbox	host	ttcpr	128mb	bytes	134217728	0.000	N/A
n2n	vbox	host	ttcpr	128mb	sec	28.195	0.200	0.071
n2n	vbox	host	ttcpr	32mb	bw	4648.668	56.538	20.232
n2n	vbox	host	ttcpr	32mb	bytes	INVALID	INVALID	INVALID
n2n	vbox	host	ttcpr	32mb	sec	6.581	1.224	0.438
n2n	vbox	host	ttcpr	64mb	bw	4658.930	35.582	12.732

Net-work	Hyper-visor	Loc-ation	Bench-mark	Factor	Exper-iment	Sample Mean	Std Dev	95% CI
n2n	vbox	host	ttcpr	64mb	bytes	INVALID	INVALID	INVALID
n2n	vbox	host	ttcpr	64mb	sec	13.129	2.432	0.870
n2n	vbox	host	ttcpr	default	bw	4660.709	58.431	20.909
n2n	vbox	host	ttcpr	default	bytes	16777216	0.000	N/A
n2n	vbox	host	ttcpr	default	sec	3.516	0.047	0.017
n2n	vbox	host	ttcpt	128mb	bw	4678.980	36.049	12.900
n2n	vbox	host	ttcpt	128mb	bytes	134217728	0.000	N/A
n2n	vbox	host	ttcpt	128mb	sec	28.015	0.216	0.077
n2n	vbox	host	ttcpt	32mb	bw	4743.242	48.717	17.433
n2n	vbox	host	ttcpt	32mb	bytes	33554432	0.000	N/A
n2n	vbox	host	ttcpt	32mb	sec	6.909	0.071	0.026
n2n	vbox	host	ttcpt	64mb	bw	4720.416	33.179	11.873
n2n	vbox	host	ttcpt	64mb	bytes	67108864	0.000	N/A
n2n	vbox	host	ttcpt	64mb	sec	13.883	0.098	0.035
n2n	vbox	host	ttcpt	default	bw	4823.769	54.329	19.441
n2n	vbox	host	ttcpt	default	bytes	16777216	0.000	N/A
n2n	vbox	host	ttcpt	default	sec	3.397	0.038	0.014
n2n	vbox	vm	ping	default	ping1	2.807	0.882	0.316
n2n	vbox	vm	ping	default	ping2	0.735	0.214	0.076
n2n	vbox	vm	ttcpr	128mb	bw	2546.520	69.183	24.756
n2n	vbox	vm	ttcpr	128mb	bytes	134217728	0.000	N/A
n2n	vbox	vm	ttcpr	128mb	sec	51.509	1.456	0.521
n2n	vbox	vm	ttcpr	32mb	bw	2689.137	144.465	51.695
n2n	vbox	vm	ttcpr	32mb	bytes	33554432	0.000	N/A
n2n	vbox	vm	ttcpr	32mb	sec	12.222	0.702	0.251
n2n	vbox	vm	ttcpr	64mb	bw	2589.402	203.164	72.700
n2n	vbox	vm	ttcpr	64mb	bytes	67108864	0.000	N/A
n2n	vbox	vm	ttcpr	64mb	sec	25.466	2.061	0.738
n2n	vbox	vm	ttcpr	default	bw	2483.974	363.344	130.018
n2n	vbox	vm	ttcpr	default	bytes	16777216	0.000	N/A
n2n	vbox	vm	ttcpr	default	sec	6.747	1.067	0.382
n2n	vbox	vm	ttcpt	128mb	bw	2551.549	71.030	25.417
n2n	vbox	vm	ttcpt	128mb	bytes	134217728	0.000	N/A
n2n	vbox	vm	ttcpt	128mb	sec	51.410	1.492	0.534
n2n	vbox	vm	ttcpt	32mb	bw	2710.288	152.979	54.742
n2n	vbox	vm	ttcpt	32mb	bytes	33554432	0.000	N/A
n2n	vbox	vm	ttcpt	32mb	sec	12.130	0.737	0.264
n2n	vbox	vm	ttcpt	64mb	bw	2598.151	208.680	74.674
n2n	vbox	vm	ttcpt	64mb	bytes	67108864	0.000	N/A
n2n	vbox	vm	ttcpt	64mb	sec	25.387	2.108	0.754
n2n	vbox	vm	ttcpt	default	bw	2515.330	386.338	138.247
n2n	vbox	vm	ttcpt	default	bytes	16777216	0.000	N/A
n2n	vbox	vm	ttcpt	default	sec	6.679	1.112	0.398
n2n	xen	host	ping	default	ping1	4.574	0.332	0.119
n2n	xen	host	ping	default	ping2	0.449	0.017	0.006

Net-work	Hyper-visor	Loc-ation	Bench-mark	Factor	Exper-iment	Sample Mean	Std Dev	95% CI
n2n	xen	host	ttcpr	128MB	bw	4408.448	18.826	6.737
n2n	xen	host	ttcpr	128MB	bytes	134217728	0.000	N/A
n2n	xen	host	ttcpr	128MB	sec	29.733	0.128	0.046
n2n	xen	host	ttcpr	32MB	bw	4397.397	18.476	6.612
n2n	xen	host	ttcpr	32MB	bytes	33554432	0.000	N/A
n2n	xen	host	ttcpr	32MB	sec	7.451	0.031	0.011
n2n	xen	host	ttcpr	64MB	bw	4395.922	18.730	6.702
n2n	xen	host	ttcpr	64MB	bytes	67108864	0.000	N/A
n2n	xen	host	ttcpr	64MB	sec	14.908	0.062	0.022
n2n	xen	host	ttcpr	default	bw	4381.373	18.291	6.545
n2n	xen	host	ttcpr	default	bytes	16777216	0.000	N/A
n2n	xen	host	ttcpr	default	sec	3.739	0.016	0.006
n2n	xen	host	ttcpt	128mb	bw	4482.822	22.005	7.874
n2n	xen	host	ttcpt	128mb	bytes	134217728	0.000	N/A
n2n	xen	host	ttcpt	128mb	sec	29.239	0.144	0.051
n2n	xen	host	ttcpt	32mb	bw	4682.886	37.381	13.376
n2n	xen	host	ttcpt	32mb	bytes	33554432	0.000	N/A
n2n	xen	host	ttcpt	32mb	sec	6.997	0.056	0.020
n2n	xen	host	ttcpt	64mb	bw	4554.588	21.631	7.740
n2n	xen	host	ttcpt	64mb	bytes	67108864	0.000	N/A
n2n	xen	host	ttcpt	64mb	sec	14.389	0.069	0.025
n2n	xen	host	ttcpt	default	bw	4771.723	59.672	21.353
n2n	xen	host	ttcpt	default	bytes	16777216	0.000	N/A
n2n	xen	host	ttcpt	default	sec	3.434	0.042	0.015
n2n	xen	vm	ping	default	ping1	2.691	0.216	0.077
n2n	xen	vm	ping	default	ping2	0.482	0.015	0.005
n2n	xen	vm	ttcpr	128mb	bw	3965.723	20.360	7.286
n2n	xen	vm	ttcpr	128mb	bytes	134217728	0.000	N/A
n2n	xen	vm	ttcpr	128mb	sec	33.052	0.171	0.061
n2n	xen	vm	ttcpr	32mb	bw	4000.860	7.911	2.831
n2n	xen	vm	ttcpr	32mb	bytes	33554432	0.000	N/A
n2n	xen	vm	ttcpr	32mb	sec	8.191	0.016	0.006
n2n	xen	vm	ttcpr	64mb	bw	3986.474	12.917	4.622
n2n	xen	vm	ttcpr	64mb	bytes	67108864	0.000	N/A
n2n	xen	vm	ttcpr	64mb	sec	16.440	0.053	0.019
n2n	xen	vm	ttcpr	default	bw	4027.671	37.374	13.374
n2n	xen	vm	ttcpr	default	bytes	16777216	0.000	N/A
n2n	xen	vm	ttcpr	default	sec	4.068	0.036	0.013
n2n	xen	vm	ttcpt	128mb	bw	3981.741	21.017	7.521
n2n	xen	vm	ttcpt	128mb	bytes	134217728	0.000	N/A
n2n	xen	vm	ttcpt	128mb	sec	32.920	0.175	0.063
n2n	xen	vm	ttcpt	32mb	bw	4065.402	11.683	4.180
n2n	xen	vm	ttcpt	32mb	bytes	33554432	0.000	N/A
n2n	xen	vm	ttcpt	32mb	sec	8.060	0.023	0.008
n2n	xen	vm	ttcpt	64mb	bw	4017.826	13.763	4.925

Net-work	Hyper-visor	Loc-ation	Bench-mark	Factor	Exper-iment	Sample Mean	Std Dev	95% CI
n2n	xen	vm	ttcpt	64mb	bytes	67108864	0.000	N/A
n2n	xen	vm	ttcpt	64mb	sec	16.312	0.056	0.020
n2n	xen	vm	ttcpt	default	bw	4161.178	52.310	18.719
n2n	xen	vm	ttcpt	default	bytes	16777216	0.000	N/A
n2n	xen	vm	ttcpt	default	sec	3.937	0.049	0.017

Bibliography

[1] M. G. W. T. Lord, "EnglishUsaf cyberspace command: To fly and fight in cyberspace," *EnglishStrategic Studies Quarterly*, vol. 2, no. 3, pp. 5–17, Fall 2008. [Online]. Available: http://www.au.af.mil/au/ssq/2008/fall/lord.pdf

[2] N. A. Donley, Michael B. Schwartz, "Air force cyberspace mission alignment," Web, August 2009. [Online]. Available: http://www.af.mil/information/viewpoints/-jvp.asp?id=498

[3] M. Zyda, "From visual simulation to virtual reality to games," *Computer*, vol. 38, no. 9, pp. 25–32, 2005.

[4] M. G. Wabiszewski, Jr., T. R. Andel, B. E. Mullins, and R. W. Thomas, "Enhancing realistic hands-on network training in a virtual environment," in *SpringSim '09: Proceedings of the 2009 Spring Simulation Multiconference*. San Diego, CA, USA: Society for Computer Simulation International, 2009, pp. 1–8.

[5] J. S. Robin and C. E. Irvine, "Analysis of the intel pentium's ability to support a secure virtual machine monitor," in *SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2000, pp. 10–10.

[6] A. Gaspar, S. Langevin, W. Armitage, and M. Rideout, "Enabling new pedagogies in operating systems and networking courses with state of the art open source kernel and virtualization technologies," *J. Comput. Small Coll.*, vol. 23, no. 5, pp. 189–198, 2008.

[7] K. L. Kroeker, "The evolution of virtualization," *Commun. ACM*, vol. 52, no. 3, pp. 18–20, 2009.

[8] C. Li, "Blur the boundary between the virtual and the real," *J. Comput. Small Coll.*, vol. 24, no. 3, pp. 39–45, 2009.

[9] B. Stackpole, J. Koppe, T. Haskell, L. Guay, and Y. Pan, "Decentralized virtualization in systems administration education," in *SIGITE '08: Proceedings of the 9th ACM SIGITE conference on Information technology education*. New York, NY, USA: ACM, 2008, pp. 249–254.

[10] P. Li, "Exploring virtual environments in a decentralized lab," *SIGITE Newsl.*, vol. 6, no. 1, pp. 4–10, 2009.

[11] B. Stackpole, "The evolution of a virtualized laboratory environment," in *SIGITE '08: Proceedings of the 9th ACM SIGITE conference on Information technology education*. New York, NY, USA: ACM, 2008, pp. 243–248.

[12] IBM, “Virtualization in education,” IBM Global Education, White Paper, October 2007. [Online]. Available: http://www-07.ibm.com/solutions/in/education/download/-Virtualization%20in%20Education.pdf

[13] C. Border, “The development and deployment of a multi-user, remote access virtualization system for networking, security, and system administration classes,” in *SIGCSE ’07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*. New York, NY, USA: ACM, 2007, pp. 576–580.

[14] R. Goldberg, “EnglishSurvey of virtual machine research,” *EnglishComputer*, vol. 7, no. 6, pp. 34–45, June 1974. [Online]. Available: https://www.cs.ucsb.edu/-~ravenben/papers/coreos/Gol74.pdf

[15] M. T. Jones, “Virtual linux: An overview of virtualization methods, architectures, and implementations,” Online, Dec 2006. [Online]. Available: http://www.ibm.com/-developerworks/library/l-linuxvirt/

[16] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford, “Trellis: a platform for building flexible, fast virtual networks on commodity hardware,” in *CONEXT ’08: Proceedings of the 2008 ACM CoNEXT Conference*. New York, NY, USA: ACM, 2008, pp. 1–6.

[17] W. Sun, V. Katta, K. Krishna, and R. Sekar, “V-netlab: an approach for realizing logically isolated networks for security experiments,” in *CSET’08: Proceedings of the conference on Cyber security experimentation and test*. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–6.

[18] M. Goldweber and R. Davoli, “Vde: an emulation environment for supporting computer networking courses,” in *ITiCSE ’08: Proceedings of the 13th annual conference on Innovation and technology in computer science education*. New York, NY, USA: ACM, 2008, pp. 138–142.

[19] C. Caini, R. Firrincieli, R. Davoli, and D. Lacamera, “Virtual integrated tcp testbed (vitt),” in *TridentCom ’08: Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1–6.

[20] VMware. (2007, November) Understanding full virtualization, paravirtualization, and hardware assist. Online. VMware. [Online]. Available: http://www.vmware.com/-files/pdf/VMware_paravirtualization.pdf

[21] IBM. (2005, December) EnglishIbm systems virtualization. Online. IBM. [Online]. Available: http://publib.boulder.ibm.com/infocenter/eserver/v1r2/topic/eicay/-eicay.pdf

[22] U. Drepper, “The cost of virtualization,” *Queue*, vol. 6, no. 1, pp. 28–35, 2008.

[23] G. J. Popek and R. P. Goldberg, “Formal requirements for virtualizable third generation architectures,” *Commun. ACM*, vol. 17, no. 7, pp. 412–421, 1974.

[24] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” in *SOSP ’03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003, pp. 164–177.

[25] J. Dike, “User-mode linux,” in *ALS ’01: Proceedings of the 5th annual Linux Showcase & Conference*. Berkeley, CA, USA: USENIX Association, 2001, pp. 2–2.

[26] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, “Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors,” in *EuroSys ’07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. New York, NY, USA: ACM, 2007, pp. 275–287.

[27] kvm. (2010, Feb) Main page - kvm. [Online]. Available: http://www.linux-kvm.org/page/Main_Page

[28] T. P. Morgan. (2007, October) Kvm developer launches as qumranet with desktop virtualization. Online. [Online]. Available: http://www.itjungle.com/tlb/tlb101607-story09.html

[29] C. Babcock. (2008, February) Sun buys innotek to build out virtualization products. Online. InformationWeek. [Online]. Available: http://-www.informationweek.com/news/software/open_source/-showArticle.jhtml?articleID=206501966

[30] Oracle. (2009, Apri) Oracle press release: Oracle buys sun. Online. Oracle. [Online]. Available: http://www.oracle.com/us/corporate/press/018363

[31] J. Watson, “Virtualbox: bits and bytes masquerading as machines,” *Linux J.*, vol. 2008, no. 166, p. 1, 2008.

[32] Q. Jia, Z. Wang, and A. Stavrou, “The heisenberg measuring uncertainty in lightweight virtualization testbeds,” in *USENIX 2nd Annual Cyber Security Experiment and Test Workshop*, 2009.

[33] B. des Ligneris, “Virtualization of linux based computers: The linux-vserver project,” in *HPCS ’05: Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 340–346.

[34] libvirt. (2010, May) libvirt: The virtualization api. Online. [Online]. Available: http://libvirt.org/

[35] N. M. K. Chowdhury and R. Boutaba, “A survey of network virtualization,” *Comput. Netw.*, vol. 54, no. 5, pp. 862–876, 2010.

[36] D. Yen, D. Havelka, and D. C. Chou, “Virtual private networks: a model for assessing alternatives,” *Int. J. Netw. Virtual Organ.*, vol. 1, no. 1, pp. 91–113, 2002.

[37] M. Tsugawa and J. A. B. Fortes, “Characterizing user-level network virtualization: Performance, overheads and limits,” in *ESCIENCE ’08: Proceedings of the 2008 Fourth IEEE International Conference on eScience*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 206–213.

[38] A. Ganguly, P. O. Boykin, D. I. Wolinsky, and R. J. Figueiredo, “Improving peer connectivity in wide-area overlays of virtual workstations,” *Cluster Computing*, vol. 12, no. 2, pp. 239–256, 2009.

[39] D. I. Wolinsky, L. Abraham, K. Lee, Y. Liu, J. Xu, P. O. Boykin, and R. J. O. Figueiredo, “On the design and implementation of structured p2p vpns,” *CoRR*, vol. abs/1001.2575, pp. 1–15, 2010.

[40] L. Deri and R. Andrews, “N2n: A layer two peer-to-peer vpn,” in *AIMS ’08: Proceedings of the 2nd international conference on Autonomous Infrastructure, Management and Security*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 53–64.

[41] D. I. Wolinsky, Y. Liu, P. S. Juste, G. Venkatasubramanian, and R. Figueiredo, “On the design of scalable, self-configuring virtual networks,” in *SC ’09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. New York, NY, USA: ACM, 2009, pp. 1–12.

[42] S. Abbott-McCune, A. J. Newtson, J. Girard, and B. S. Goda, “Developing a reconfigurable network lab,” in *SIGITE ’08: Proceedings of the 9th ACM SIGITE conference on Information technology education*. New York, NY, USA: ACM, 2008, pp. 255–258.

[43] B. E. Mullins, T. H. Lacey, R. F. Mills, J. M. Trechter, and S. D. Bass, “The impact of the nsa cyber defense exercise on the curriculum at the air force institute of technology,” in *HICSS ’07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*. Washington, DC, USA: IEEE Computer Society, 2007, p. 271b.

[44] V. J. H. Powell, C. T. Davis, R. S. Johnson, P. Y. Wu, J. C. Turchek, and I. W. Parker, “Vlabnet: the integrated design of hands-on learning in information security and networking,” in *InfoSecCD ’07: Proceedings of the 4th annual conference on Information security curriculum development*. New York, NY, USA: ACM, 2007, pp. 1–7.

[45] B. R. Anderson, A. K. Joines, and T. E. Daniels, “Xen worlds: leveraging virtualization in distance education,” in *ITiCSE ’09: Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education*. New York, NY, USA: ACM, 2009, pp. 293–297.

[46] ATC. (2010, May) Cydest - cyber defense trainer. Online. ATC. [Online]. Available: http://www.atcorp.com/Products/cydest.html

[47] M. Maheswaran, A. Malozemoff, D. Ng, S. Liao, S. Gu, B. Maniymaran, J. Raymond, R. Shaikh, and Y. Gao, “Gini: a user-level toolkit for creating micro internets for teaching & learning computer networking,” in *SIGCSE ’09: Proceedings of the 40th ACM technical symposium on Computer science education*. New York, NY, USA: ACM, 2009, pp. 39–43.

[48] H. Owen, “Georgia tech "hands on" network security laboratory,” Georgia Institute of Technology, Tech. Rep., 2004.

[49] J. M. D. Hill, C. A. Carver, Jr., J. W. Humphries, and U. W. Pooch, “Using an isolated network laboratory to teach advanced networks and security,” *SIGCSE Bull.*, vol. 33, no. 1, pp. 36–40, 2001.

[50] T. Winters, R. Ausanka-Crues, M. Kegel, E. Shimshock, D. Turner, and M. Erlinger, “Tinkernet: a low-cost and ready-to-deploy networking laboratory platform,” in *ACE ’06: Proceedings of the 8th Austalian conference on Computing education*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 253–259.

[51] W. D. Armitage, A. Gaspar, and M. Rideout, “Remotely accessible sandboxed environment with application to a laboratory course in networking,” in *SIGITE ’07: Proceedings of the 8th ACM SIGITE conference on Information technology education*. New York, NY, USA: ACM, 2007, pp. 83–90.

[52] J. P. Walters, V. Chaudhary, M. Cha, S. G. Jr., and S. Gallo, “A comparison of virtualization technologies for hpc,” in *AINA ’08: Proceedings of the 22nd International Conference on Advanced Information Networking and Applications*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 861–868.

[53] proxmox. (2010, Feb) Proxmox ve. [Online]. Available: http://pve.proxmox.com/-wiki/Main_Page

[54] A. Ganguly, A. Agrawal, P. Boykin, and R. Figueiredo, “Ip over p2p: enabling self-configuring virtual ip networks for grid computing,” in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 25-29 2006, p. 10 pp.

[55] phoronix. (2010, Feb) Phoronix test suite - linux testing & bechmarking platform. [Online]. Available: http://www.phoronix-test-suite.com/

[56] S. Bratanov, R. Belenov, and N. Manovich, “Virtual machines: a whole new world for performance analysis,” *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 46–55, 2009.

[57] V. Forums. (2009, June) Code complilation in guests slow vs native/kvm/esxi. Online. Oracle. [Online]. Available: http://78.46.147.154/-viewtopic.php?f=1&t=16886&start=0

[58] openvz. (2010, Feb) Openvz - wiki. [Online]. Available: http://wiki.openvz.org/-Main_Page

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
17-06-2010	Master's Thesis	Aug 2009 - June 2010

4. TITLE AND SUBTITLE
Designing a Hybrid Virtualization Platform Design for Cyber Warfare and Simulation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Stewart, Kyle E., 2d Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCE/ENG/10-06

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
INTENTIONALLY LEFT BLANK

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Students studying topics in cyber security benefit from working with realistic training labs that test their knowledge of network security. Cost, space, time, and reproducibility are major factors that prevent instructors from building realistic networks for their students. This research explores the ways that existing virtualization technologies could be packaged to provide a more accessible, comprehensive, and realistic training and education environment. The research will look into ways of merging two existing virtualization methods in order to leverage the unique benefits that each type of virtualization techniques provides. The first method, called operating system virtualization, provides a highly memory efficient way to run virtual machines, provided that all virtual machines run the same operating system kernel. Full virtualization requires a larger memory footprint, but allows arbitrary operating systems to be virtualized. Combining these two techniques will allow for larger, more diverse training environments for modeling and training in the cyber domain.

15. SUBJECT TERMS
Cyber Warfare, Cyber Security, Virtualization, Computer Networks

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
REPORT	**ABSTRACT**	**c. THIS PAGE**			Jeffrey W. Humphries, Lt Col, USAF (ENG)
U	**U**	**U**	**UU**	111	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, x7253 jeffrey.humphries@afit.edu